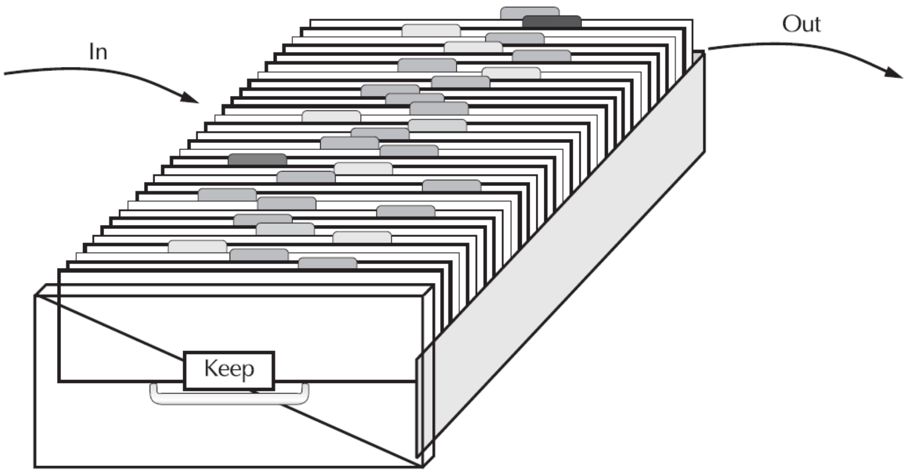
ORACLE

Oracle Database 12c is a significant upgrade from prior releases of Oracle. New features give developers, database administrators, and end users greater control over the storage, processing, and retrieval of their data.

A relational database management system (RDBMS) such as Oracle gives you a way of doing these tasks in an understandable and reasonably uncomplicated way.

i) Lets you put data into it Keeps the data
ii) Lets you get the data out
iii) and work with it

Oracle supports this in/keep/out approach and provides clever tools that allow you considerable sophistication in how the data is captured, edited, modified, and put in; how you keep it securely; and how you get it out to manipulate and report on it.



Oracle is a RELATIONAL DATA BASE MANAGEMENT SYSTEM (RDBMS)
Oracle data base is a software where we can store and process (fetch / insert / change / delete) business data.
Oracle is an RDBMS s/w from oracle corp.

How the data is generated?

Through business objects and its activities (transactions), data will be generated.

What is a business?
Business is a collection of real world entities and its activities.

```
                              BUSINESS
                                |
--------------------------------------------------------------------------------------------
|             |              |                     |
{emps         depts          Products/             customers}-----------> [entities/ Objects]
                             services
```

In early days of a business, we have very limited data like

- limited number of employees
- limited departments
- limited products
- limited customers (No customers on the very beginning day)

Day-By-Day the business may improve, means need more number of business resources and there is an increase in number of transactions. In this case we need the automated Business system called DBMS, to maintain business data and its activities automatically.

DATA BASE CONCEPTS:

DATA:

Collection of information of any one Business entity is known as data.
[One line of information]

Ex: one employee information
One product information
One sales transaction information

DATA BASE:
It is software which stores and manages the collection of information of all objects in the business.
Technically, it is collection of programs and each program is responsible for performing a specific task.

DATABASE MANAGEMENT SYSTEM:
DB which is comprised with management system services is known as DBMS.
Here the services are

- Entering new data
- Updating old data with new data
- Deleting unwanted data
- Authenticating the users
- Providing security.

RDBMS:
Author of RDBMS IS E.F. CODD and he invented 12 Rules for an RDBMS.
Collection of interrelated data of all interrelated objects with in the business is known as RDBMS.
The relation between the tables is implemented by using Referential integrity constraints.

In any **RDBMS**

- Data stored in the form of 2-dimensional tables
- A table is a collection of rows and columns.
- A row is known as record (collection of columns)
- A column is known as a field

Tables of Information

Oracle stores information in tables. Each of these tables has one or more columns. The informationis stored row after row. Each unique set of data gets its own row.
Oracle avoids specialized, academic terminology in order to make the product more approachable. In research papers on relational theory, **a column** may be called an **"attribute,"** a **row** may be called a **"tuple"** and a **table** may be called an **"entity."**
For an end user, however, these terms are unnecessary.

Ex: WITHOUT RELATION

We are unable to fetch some kind of required data. Consider the below example. By maintaining data in 2 individual tables [no relation between them] we are unable to fetch relevant data from these tables.

Try to find the answers for below questions.
1) Number of products from Sony?
2) Company details of product id "p001"?
3) Total investment made for Asus products?......etc.

Prod_Dtls

PID	PNAME	COST	MFG	WARRENTY
P001	Mobile	14000	22-oct-14	1 year
P003	Desktop	27000	14-may-15	3 years
P006	Laptop	35990	11-may-12`	2 years
P004	Tablet	12000	21-mar-13	1 year
P005	Smart phone	37000	10-oct-15	1 year

Comp_Dtls

Comp_code	Comp_name	Country
Cmp1	Sony	Japan
Cmp2	Asus	South Korea

Ex**: WITH RELATION**

By using **Primary key** of one table we need to implement **foreign key** in other table. This is called as implementing Physical relation between the tables.

See Below example:

Comp_Dtls

Comp_code	Comp_name	Country
Cmp1	Sony	Japan
Cmp2	Asus	South Korea

Prod_Dtls

Pid	Pname	Cost	Mfg	Warrenty	Comp_Code
P001	Mobile	14000	22-oct-14	1 year	Cmp1
P003	Desktop	27000	14-may-15	3 years	Cmp2
P006	Laptop	35990	11-may-12`	2 years	Cmp1
P004	Tablet	12000	21-mar-13	1 year	Cmp1
P005	Smart phone	37000	10-oct-15	1 year	Cmp2

Ex: Information for below requirement is easy now.

Find Number of products from Sony?
Find Number of products from Asus?
Average product cost from any company?
Company details of any product?

Advantages:

We can maintain integrity among table data
We can fetch accurate and complete data.

E.F. CODD Rules

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

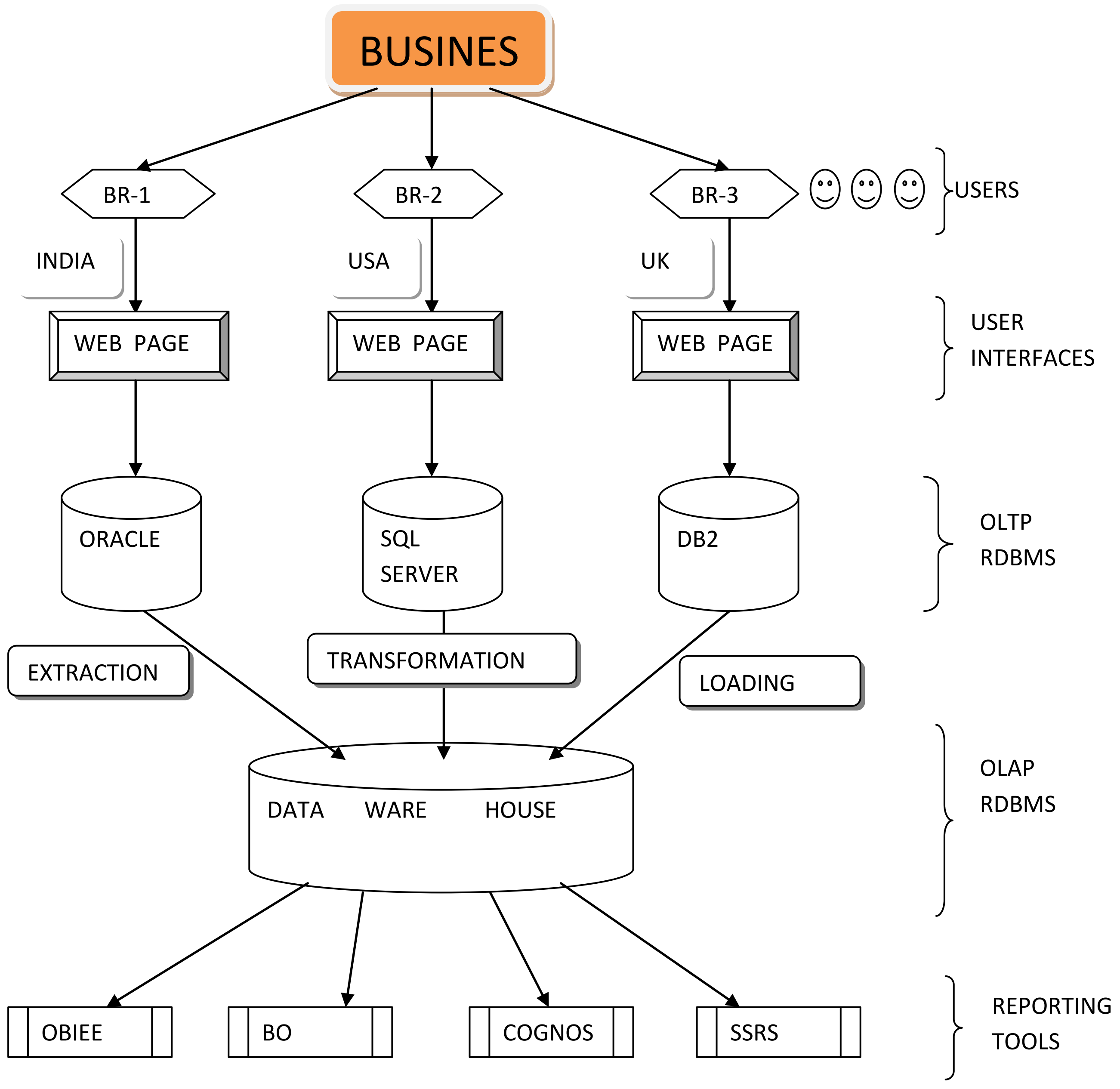
Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

PROJECT DEVELOPMENT PROCESS



OLTP RDBMS: **On Line Transaction Processing**

- It contains transactional data/ day-to-day data/ current data / dynamic data
- It is used to store or process the business data.

OLAP RDBMS: **On Line Analytical Processing**

- It contains historical data / old data.
- It is used to analyze the business.

Brief History of Oracle Database

The current version of Oracle Database is the result of over 35 years of innovative development.

The current version of Oracle Database is the result of over 30 years of innovative development. Highlights in the evolution of Oracle Database include the following:

- Founding of Oracle

  In 1977, Larry Ellison, Bob Miner, and Ed Oates started the consultancy Software Development Laboratories, which became Relational Software, Inc. (RSI). In 1983, RSI became Oracle Systems Corporation and then later Oracle Corporation.

- First commercially available RDBMS

  In 1979, RSI introduced Oracle V2 (Version 2) as the first commercially available **SQL**-based RDBMS, a landmark event in the history of relational databases.

- Portable version of Oracle Database

  Oracle Version 3, released in 1983, was the first relational database to run on mainframes, minicomputers, and PCs. The database was written in C, enabling the database to be ported to multiple platforms.

- Enhancements to concurrency control, data distribution, and scalability

  Version 4 introduced multi-version **read consistency**. Version 5, released in 1985, supported client/server computing and **distributed database** systems. Version 6 brought enhancements to disk I/O, row locking, scalability, and backup and recovery. Also, Version 6 introduced the first version of the **PL/SQL** language, a proprietary procedural extension to SQL.

- PL/SQL stored program units

  Oracle7, released in 1992, introduced PL/SQL stored procedures and triggers.

- Objects and partitioning

  Oracle8 was released in 1997 as the object-relational database, supporting many new data types. Additionally, Oracle8 supported partitioning of large tables.

- Internet computing

  Oracle8*i* Database, released in 1999, provided native support for internet protocols and server-side support for Java. Oracle8*i* was designed for internet computing, enabling the database to be deployed in a multitier environment.

- Oracle Real Application Clusters (Oracle RAC)

Oracle9*i* Database introduced Oracle RAC in 2001, enabling multiple instances to access a single database simultaneously. Additionally, Oracle XML Database (**Oracle XML DB**) introduced the ability to store and query XML.

- Grid computing

  Oracle Database 10*g* introduced **grid computing** in 2003. This release enabled organizations to virtualize computing resources by building a **grid infrastructure** based on low-cost commodity servers. A key goal was to make the database self-managing and self-tuning. **Oracle Automatic Storage Management (Oracle ASM)** helped achieve this goal by vitalizing and simplifying database storage management.

- Manageability, diagnoisability, and availability

  Oracle Database 11*g*, released in 2007, introduced a host of new features that enabled administrators and developers to adapt quickly to changing business requirements. The key to adaptability is simplifying the information infrastructure by consolidating information and using automation wherever possible.

- Plugging In to the Cloud

  Oracle Database 12*c*, released in 2013, was designed for the Cloud, featuring a new Multitenant architecture, In-Memory column store, and support for JSON documents. Oracle Database 12*c* helps customers make more efficient use of their IT resources, while continuing to reduce costs and improve service levels for users.

Oracle Database Architecture

A **database server** is the key to information management. In general, a **server** reliably manages a large amount of data in a multiuser environment so that users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Database and Instance

An Oracle database server consists of a database and at least one **database instance**, commonly referred to as simply an *instance*. Because an instance and a database are so closely connected, the term **Oracle database** is sometimes used to refer to both instance and database.

- Database

  A database is a set of files, located on disk, that store data. These files can exist independently of a database instance.

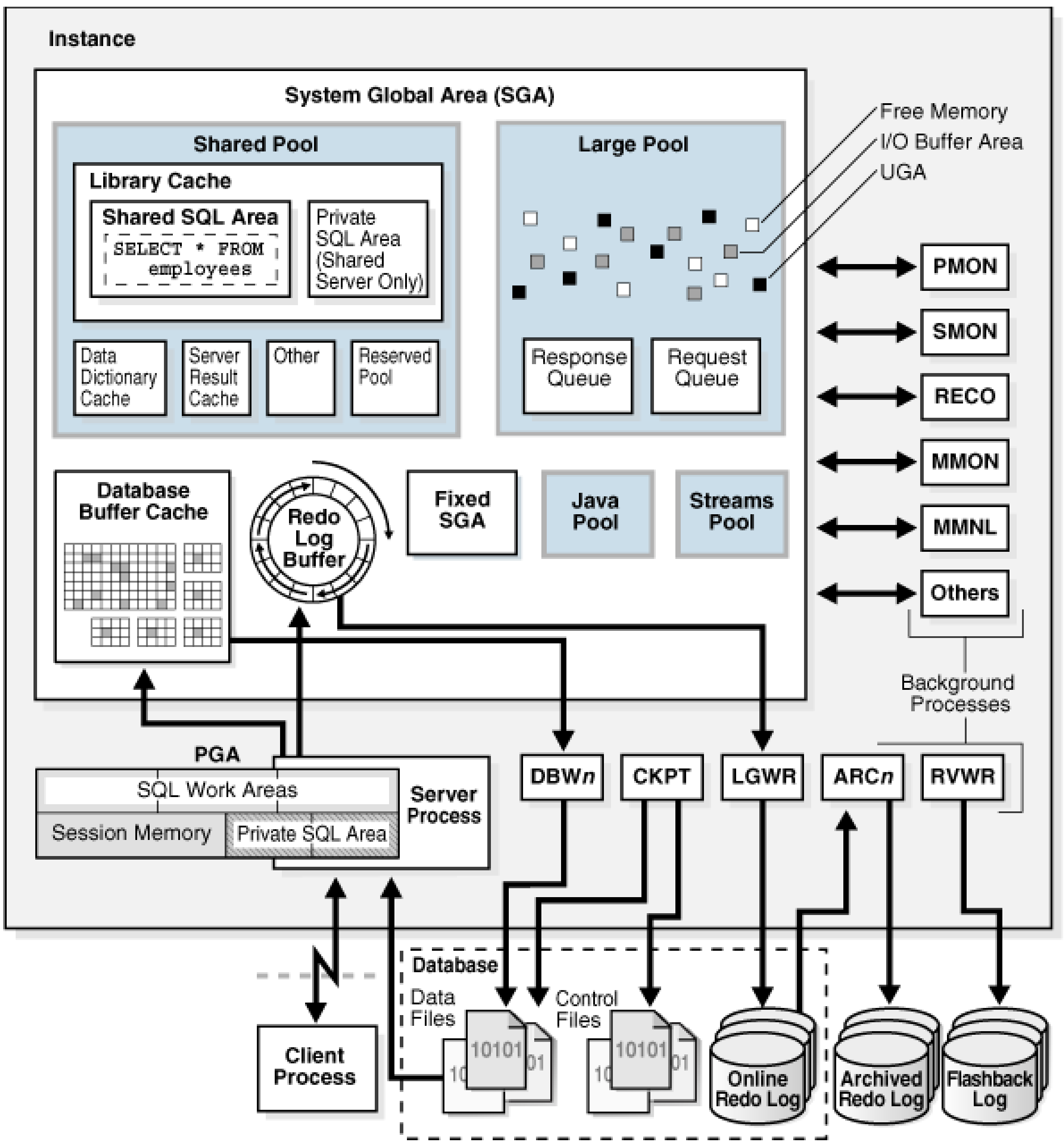
- Database instance

  An instance is a set of memory structures that manage database files. The instance consists of a shared memory area, called the **system global area (SGA)**, and a set of background processes. An instance can exist independently of database files.

Figure 1-1 shows a database and its instance.

- For each user connection to the instance, a **client process** runs the application.
- Each client process is associated with its own **server process**.
- The server process has its own private session memory, known as the **program global area (PGA)**.

Oracle Instance and Database



A database can be considered from both a physical and logical perspective. Physical data is data viewable at the operating system level. For example, operating system utilities such as the Linux **ls** and **ps** can list database files and processes. Logical data such as a table is meaningful only for the database. A SQL statement can list the tables in an Oracle database, but an operating system utility cannot.

The database has physical structures and logical structures. Because the physical and logical structures are separate, you can manage the physical storage of data without affecting access to logical storage structures. For example, renaming a physical database file does not rename the tables whose data is stored in this file.

Types of users: 2

We can devide the users of data base as follows.

i) **Non-Technical user:**
These users interact with any database through GUI applications or web applications (pages).

ii) **Technical user:**
These users interacts with any database through any client tool SQL * PLUS window or any GUI (Graphical User Interface) Data Base tools like **SQL DEVELOPER, PL/SQL DEVELOPER and TOAD.**
Technical users are Developers and DBA.

Parts of Oracle

1) SQL
2) PL/SQL

SQL (Structured query language)

Oracle was the first company to release a product that used the English-based *Structured Query Language,* or *SQL.* This language allows end users to extract information themselves, without using a systems group for every little report.

Oracle’s query language has structure, just as English or any other language has structure. It has rules of grammar and syntax, but they are basically the normal rules of careful English speech and can be readily understood.

It is known as data base language. It is used to communicate with any database. We can use this language constructs to write SQL QUERIES.

SQL * PLUS is a default client tool and acts as an interface between client and database.

SQL Def:

It is a collection of pre defined commands and constructs with syntactical rules.

$$\begin{tikzcd}
\text{Client} \arrow[r, "\text{Request}", "\text{Query}"'] & \text{SQL} \arrow[r, "\text{processed on}"] & \text{DB} \arrow[dl] \\
& \text{Output} &
\end{tikzcd}$$

1. Sql is a client tool to interact with ORACLE DB /any DB
2. Sql is to be installed in to the system whenever we have installed the db software.
3. Client [Technical] requests should be submitted in the form of "Queries".
4. Queries are executed by SQL STMT EXECUTOR (Oracle Db Engine)
5. Queries are executed against database and output will be displayed on the Sql * plus window.

Features of Sql * Plus:

- At a time only one query is allowed to execute
- Sql queries are not case sensitive
- Each query is terminated with **; (semi colon)**
- SQL commands are ANSI standard (American National standard institute)

SQL COMMANDS

Types of SQL commands:

1) **DDL** (data definition language) commands:

Used to create or change or delete any data base objects

CREATE ALTER DROP TRUNCATE

RENAME

2**) DML** (data manipulation language) COMMANDS

Used to fetch data / enter new data/ changing existed data / deleting the data from table.

INSERT UPDATE DELETE TRUNCATE

3) **DRL** (data retrieval language) Command

SELECT (logical command)

4) **DCL** (DATA CONTROL LANGUAGE) COMMANDS

Used to control the access of data base objects. These commands are used by DBA (database administrator)

GRANT REVOKE

5) **TCL** (TRANSACTION CONTROL LANGUAGE) COMMANDS

Used to save or cancel the actions/transactions made on table data.

COMMIT ROLLBACK SAVEPOINT

HOW TO CREATE A USER ACCOUNT?

DBA (Data Base Administrator) can CREATE and DELETE and manage user accounts

NOTE: User name is not case sensitive, but password is case sensitive in oracle 11g

General DBA credentials

1) In login window, submit the following details

username:	system
password:	manager
Hoststring:	orcl / oracle -----> It is the database service name.

2) Now it opens Sql * Plus window

In this window, the DBA has to write queries to create user account.

Syntax: creating a new user account

Create User <user_name> Identified by <password >;

Ex: create user ***dinesh*** identified by ***welcome***;

User created.

3) Giving permissions to use the resources of data base

GRANT resource, connect to <USER_NAME>;

Ex: grant resource,connect to ***dinesh***;

Grant succeeded.

Ex: Sql> exit; [Disconnecting from database]

Ex: Connecting to data base as ***Dinesh.***

username:	dinesh
password:	welcome
Hoststring:	orcl

LOG ON PROCEDURE:

1) Double click on sqlplus icon on desktop
(In oracle 8i/9i/10g)

Or
Double click SqlPlus Command window

2) It opens a log on window, in this type the following

username:	pv
password:	welcome
hoststring:	oracle / orcl

In Oracle 11g Command window, type as below.
Enter Username: dinesh
Password : welcome

3) It opens Sql * Plus window with prompt

```
SQL>_
```

LOG OUT PROCEDURE:

In the sql window, type the command EXIT. It disconnects the current user from the data base and it close the sql window.

Ex: `SQL>EXIT;`

NAVIGATION:

START--> PROGRAMS--> ORACLE ora_Home1_11G--> Application development--> select Sql command

Then it opens Sql command window, then submit user credentials.

DATA MODEL

For any OLTP RDBMS the data model is **E-R** (Entity-Relationship) Model.
Create an Entity-Relationship (ER) model is to visually represent the structure of a business database, where data equates to entities (or objects) that are linked by defined relationships expressing dependencies and requirements. By nature it is an abstract visualization, the first step in the design process towards creating a logical and functional database.

Entity :	Any real time object is known as entity
Ex :	emp, dept, orders, transactions, sales, products, service, customers

- For each object in the business we need to create a table.
- Every object is having some Properties.
- For each property we need to maintain a column.

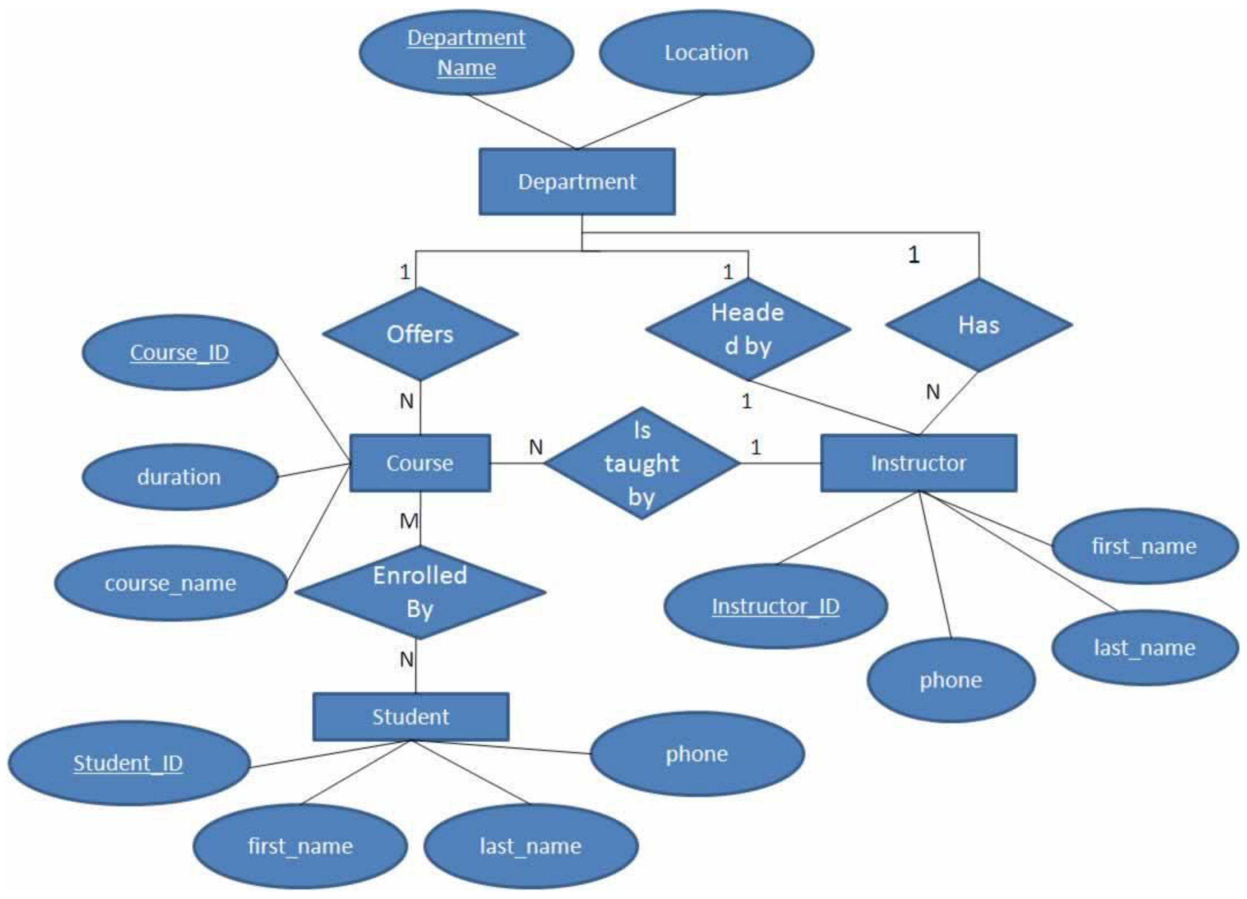
Ex : Consider emp object

Then table name is EMP
Properties of emp are as follows.

eid ename sal job hiredate gender mailid mobile ......

Relation:	How business activity is working between 2 objects
Ex :	emp<-->dept;
	company<-->product ;
	prod<-->sales ;

Sample E-R Model is as follows



DDL command

1) **CREATE**
It is used to create any data base object like tables, views, indexes, sequences, synonyms, users, functions, procedures, triggers, packages and so on.

HOW TO CREATE A TABLE?

```
syn:    CREATE  TABLE <table_name>
        (
        <colname-1> DATATYPE (size),
        <colname-2> DATATYPE (size),
        :       :       :       :,
        :       :       :       :,
        );
```

Naming Rules

Rules to follow before specifying names (Object names, Column Names and Variable Names).

i) Each name should begins alphabet
ii) Valid character set is
a-z,A-Z,0-9,@,$,# and _ (underscore)

Ex: Emp123
Emp_o11

iii) Names are not case sensitive

iv) Already existed names are not allowed

v) Pre defined keywords (Reserved Words) are not allowed as names.

vi) Blank space within a name is not allowed

vii) Max length of any name is 32 chars

Ex:

valid_names	invalid_names
prod_dtls	prod dtls--Since blank space within the name
emp@sal	emp.sal----Since " . " is not valid character
emp123	123emp-----Since Name is not beginning with alphabet
emp_info	emp-info---Since " -" is not valid char.
emp_table	table------Since " table " is a reserved word

DATATYPES:

The data type represents the type of data to be entered into a column and Db engine can assign memory for the value entered into the column.

I) String Data types:

These data types support alphabets, digits and any symbol from keyboard.

1) CHAR (size)
It is used to store fixed length character strings. By default the size is 1 character, and max size is 2000 chars or bytes.

Ex: empid, pnr number, bank account numbers, Policy Numbers and so on

2) VARCHAR2 (size)
It is used to store variable length character strings. No default size. we should specify size and max size is 4000 chars /bytes.

Ex:	emp names, addresses, descriptions, city names,

3) LONG
It is used to store variable length char data (similar to *varchar2 data type)* but max size **is 2 GB**
NOTE: Only one long type column is allowed per a table.

II) Numeric Data types

1) **NUMBER (Precision, [Scale])**
It is used to store numbers along with decimal point.
2) **NUMBER (Precision**
It is used to store numbers without decimal point.

Precision represents total number of digits in the value.
Scale represents the max number of digits after decimal point.
Total number of digits in the value should be less than or equal to Precision value.
Total number of digits in the decimal part should be less than or equal to scale value.

Note:	The max value for precision is 38

```
Ex:     prod_price      number(7,2)

        12.75
        123.1
        45621.08
        99999.99
        125

Ex:     123.4567--> Invalid
        0.0972----> Invalid

Ex:     emp_sal         number(6)

        12560
        100
        0
        10
        999999
```

III) Date data type

DATE
It is used to store date type data. Oracle is having a predefined date format as follows.

DD-MON-YY

DD	Digits of date
MON	First 3 chars of month name

YY	Last 2 digits of year

Ex: 12-may-13
Ex: 12/may/13 --It is not considered as a date format--

IV) <u>Binary Data types</u>

1) RAW (size)

It is used to store binary data like images, thumb impressions, logos and so on.
Max size is 2000 bytes < 2 KB

2) LONG RAW

It is similar to RAW data type but max size is 2 GB

NOTE: Only one long raw type column is allowed per a table.

V) LOB--Large Objects

It is used to store higher volumes of data and max size is 4 GB.

LOB TYPES -- 3

CLOB--CHAR LOB--used to store character data
BLOB-- BINARY LOB-- Used to store binary data
NCLOB--Fixed length multi char large objects--Used to store both binary and char data

---

Ex:
Write a query to create emp_info table with columns eid, ename, sal, jdate, desg, and gender with appropriate data types.

```
CREATE TABLE EMP_INFO
(
EID          NUMBER (4),
ENAME        VARCHAR2 (20),
SAL          NUMBER (5),
JDATE        DATE,
DESG         VARCHAR2 (20),
GENDER       CHAR
);
```

HOW TO INSERT DATA INTO A TABLE?

INSERT:

It is used to insert new records in to the table.

Syntax:

```
INSERT INTO <table_name>[(col1, col2,...., col n)]
VALUES (val1, val2,.....,val n);
```

Note:

i) If number of columns in the table and number of values inserting in to the table are equal, then no need to specify column names while inserting records.
ii) Char, Varchar2 and date type values should be enclosed in Single Quotes.
iii) If Number of inserting values are less than the number of columns then we must specify column names while inserting records.

Example Insertions:

a)
```
insert into emp_info (eid,ename,sal,jdate,desg,gender)
values (1111,'Dinesh',75000,'23-may-14','developer','M');
```
b)
```
insert into emp_info(eid,ename,sal,jdate,desg,gender)
values (1112,'Madhu',30000,'23-may-14','developer','F');
```
c)
```
insert into emp_info(eid,ename,sal,jdate,desg,gender)
values(1191,'Xavier',10000,'02-feb-10','clerk','M');
```

d)
```
insert into emp_info
values(1110,'john',11700,'23-may-14','developer','M');
```

e)
```
insert into emp_info
Values(1121,'dilroop',61000,'21-oct-13','developer','F');
```

f)
```
insert into emp_info
Values (1120,'abhi',10000,'23-may-14','admin','M');
```
g)
```
Insert into emp_info
Values('kiran','5000');
```

Error: Not enough values

h)
```
Insert into emp_info (ename,sal)
Values('kiran','5000');
```

Inserting records continuously using & operator:

& is known as "address operator". It indicates the address of column within the data base.

Ex:
```
Insert into emp_info
Values('&eno', '&ename', '&sal', '&jdate', '&desg', '&gender');


enter value for eno:    232
enter value for ename:          hari
enter value for sal:    23000
enter value for jdate:  22-oct-11
enter value for desg:   admin

1 row created.
```

In sql * plus window

```
sql> /  [ to re-execute the recent Query ]

enter value for eno:    231
enter value for ename: samuel
enter value for sal:    12000
enter value for jdate:  10-may-14
enter value for desg:   admin

1 row created.


sql > /
```

NULL VALUES:

A missed value in a column is known null value.
Null value is not equal to zero or space or other null values.

INSERTING NULL VALUES: 2 Methods

1) Implicit insertion:

If we miss a value in a column then it is dynamically maintains a null value at that place. To insert like this, we need to specify column names while using insert command.

Ex:
```
Insert into emp_info (eno,ename,sal,desg)
Values(555,'john',12000,'salesman');
```

2) Explicit insertion:

Just specify NULL keyword at the place of missed values.
Here no need to maintain column names while inserting data.

Ex:
```
Insert into emp_info
Values (888,'martin',12000,'salesman',null,null,NULL,Null);
```

DESCCRIBE / DESC
It is used to display table structure. A table structure contains column names, data types and sizes.

SYNTAX

DESC <Table_Name>;

EX: ***describe*** emp_info;

or

desc emp_info;

HOW TO DISPLAY LIST OF TABLE NAMES?

EX: SELECT * FROM TAB;

<u>DEFAULT</u> Keyword

We can create a table with DEFAULT Clause. While inserting data in to table , we don't need to enter any value in to default column. The column is populated with the value supplied along with DEFAULT keyword.

Syntax:

```
CREATE TABLE  < table_name>
(
Col1            datatype(size)  DEFAULT  'Value'    ,
"                       "                           ,
"                       "                           ,
);
```

Example:

```
Create Table customer
(
Cno             number(2),
Cname           varchar2(20),
City            varchar2(10)  DEFAULT  'Hyderabad'
);
```

Example: Inserting values in to Customer table

```
Insert into customer(cno,cname) values
(1,'kiran');
Insert into customer(cno,cname) values
(2,'Madhu');
Insert into customer values
(3,'dinesh',Null);
Insert into customer values
(4,'john','Texas);
```

Ex: Selecting data from Customer table.

Select * from tab;

```
1       Kiran           Hyderabad
2       Madhu           Hyderabad
3       Dinesh          [       ]
```

```
4         john              Texas
```

Note:-
By Default the column value is “Hyderabad”.
If we submit a different value or Null value then the default column is populated with given value.

<u>SELECT</u>

It is used to retrieve a logical copy of required data from a table or columns.

```
Syntax:       SELECT col1, col2,...., coln / *
              FROM  table_name;
```

Ex: Display employee names?

select *ename* from *emp_info*;

<u>OutPut:</u>

```
ENAME
--------------------
dinesh rao
smitha panday
madhu
madhu
Allen
king
torjan
john
martin

 9 rows selected
```

Ex: Get emplyee id, names and salaries?

select *eno, ename, sal* from *emp_info;*

<u>output:</u>

```
 ENO         ENAME             SAL
---------- --------------------  ----------
       111 dinesh rao            34000
       222 smitha panday         44000
       333 madhu                 20000
       333 madhu                 20000
       444 Allen                 34000
       555 king                  21000
       666 torjan                11000
       777 john                  12000
       999 martin                12000
 9 rows selected
```

Ex:	Display column data in user required order (Instead of Physical order)

	Display employee join_dates, ename, designitions and salaries?

	select *join_date, ename, desg,sal* from *emp_info*;

output:

```
JOIN_DATE ENAME               DESG                  SAL
--------- ------------------- ------------------- ----------
12-MAY-10 dinesh rao          developer             34000
12-MAY-10 smitha panday         developer             44000
21-OCT-11 madhu              admin                20000
21-OCT-11 madhu              admin                20000
              Allen              developer             34000
16-OCT-12 king              salesman              21000
11-MAY-12 torjan             clerk               11000
      john              salesman              12000
      martin             salesman              12000

 9 rows selected
```

Ex:	display employee details?

	select * from emp_info;

output:

```
ENO           ENAME              SAL          DESG           JOIN_DATE GENDER
MOBILE        MAILID
---------- ------------------- ---------- ------------------- --------- ------- ------------------------------------
    111 dinesh rao            34000 developer           12-MAY-10 male    7878787878
dinesh@hotmail.com
    222 smitha panday           44000 developer           12-MAY-10 female  7878787878
smith_p@hotmail.com
    333 madhu                20000 admin              21-OCT-11 male    5656565656
madhu@yahoo.com
    333 madhu                20000 admin              21-OCT-11 male    5656565656
madhu@yahoo.com
    444 Allen              34000 developer                 male    8989898989
    555 king              21000 salesman            16-OCT-12 male    7867676767
    666 torjan             11000 clerk             11-MAY-12 male    1111111111
t@yahoo.com
    777 john              12000 salesman
    999 martin              12000 salesman

 9 rows selected
```

<u>DISPLAYING COLUMN DATA WITH USER DEFINED TITLES:</u>

```
Syntax:        select colname "title", colname "title"...
               from table;
```

```
Ex:     select ename "Employee Name", desg " Job of Employee"
        from employee_info;
```

```
employee name        job of employee
------------------- -------------------
dinesh rao          developer
smitha panday       developer
madhu               admin
madhu               admin
Allen               developer
king                salesman
torjan              clerk
john                salesman
martin              salesman

 9 rows selected
```

Ex:

```
select mobile "Phone_number",desg "Job Title",eno "EmpId",ename "EmpNames" from
emp_info

select mobile Phone_number,desg JobTitle,eno EmpId,ename "EmpNames" from emp_info

select mobile As Phone_number,desg As JobTitle,eno AS  EmpId,ename as "EmpNames"
from emp_info
```

Assignments:

i) Create a table Item_dtls (Electronics)

ii) Create a table Sales_dtls

iii) create a table manufacturers

iv) Try to insert at least 10 records in the above tables

v) Try to insert at least 2 records with null values

<u>Clauses:</u>

DISTINCT clause

ORDER BY clause

Ex: table and records:

```
create table departments
(
did number(2),
dname  varchar2(20),
city   varchar2(20)
);

insert into departments
values(10,'Production','Dallas');
insert into departments
values(10,'Production','Dallas');
insert into departments
values(10,'Production','Dallas');
insert into departments
values(10,'Production','Dallas');
insert into departments
values(10,'Production','Dallas');
insert into departments
values(10,'Production','Dallas');

insert into departments
values(20,'Sales','Texas');
insert into departments
values(20,'Sales','Texas');
insert into departments
values(20,'Sales','Texas');
insert into departments
values(20,'Sales','Texas');

insert into departments
values(30,'Finance','Chicago');
insert into departments
values(30,'Finance','Chicago');
insert into departments
values(30,'Finance','Chicago');
insert into departments
values(30,'Finance','Chicago');
insert into departments
values(30,'Finance','Chicago');
```

DISTINCT Clause:

It will display different / unique values from the column and it will also display unique records from the table.

Syntax: select distinct (colname) from <table>;

Syntax: select distinct colname1,colname2,.... from <table>;

Ex: display list of different designitions?

```
select distinct(desg) " list of jobs"  from emp_info;
```

```
list of jobs
--------------------
salesman
developer
clerk
admin
```

Ex: display distinct department names?

```
select distinct(dname) from dept;
```

output:

```
        Production
        Sales
        Finance
```

Ex: Get unique records from the above table?
or
Display records from the above table without duplicates?

```
select distinct did,dname,city  from dept;
```

```
10      production      chicago
20      Sales           Texas
30      Finance                 Dallas
```

ORDER BY Clause:

It is used to display the column data or table data in ascending / descending or sorting [a to z] / reverse sorting [z-a] data.

Syntax: select *col1,..... / **
from *TableName*
order by *col1, col2,.....* [asc / desc];

Note:

i) By default it will display a to z data or ascending order data.
ii) In case of char data, if there exists upper and lower case data then order by gives highest priority to the upper case data and next priority goes to lower case data.

Since upper case data A-Z ascii values are 65-90
lower case data a-z ascii values are 91-122

Ex: Display employee names in alphabetical (sorting) order?

select *ename* from *emp* **ORDER BY** *ename*;

sample output:

```
ENAME
----------
ADAMS
ALLEN
BLAKE
CLARK
FORD ...
```

Ex: Display employee names in reverse order?

select *ename* from *emp* **ORDER BY** *ename* **DESC**;

sample output:

```
ENAME
----------
WARD
TURNER
SMITH
SCOTT
MILLER ...
```

Ex: display ename,sal, desg on the order of salary?

select *ename, sal, job* from *emp* **order by** *sal;*

sample output:

```
ENAME      SAL                    JOB
---------- --------------------- ---------
SMITH 800             CLERK
JAMES 950              CLERK
ADAMS 1100             CLERK
WARD 1250             SALESMAN
MARTIN 1250             SALESMAN
```

<u>***ORDER BY*** clause on more than one column:</u>

Here the first priority of order by clause given to first column, if first column having duplicates then order by priority goes to second column and so on.

Ex: display sal and ename from emp based on order of salaries?

select sal,ename from emp order by sal;

sample output:

```
SAL                   ENAME
--------------------- ----------
```

```
800       SMITH
950       JAMES
1100      ADAMS
1250      WARD------->
1250      MARTIN ---->
1300      MILLER
1500      TURNER
1600      ALLEN
2450      CLARK
2850      BLAKE
2975      JONES
3000      SCOTT---->
3000      FORD ---->
5000      KING
```

Ex: display sal and ename from emp based on order of salaries and employee names?

```
select sal,ename from emp order by sal,ename;
```

sample output:

```
SAL          ENAME
---------------------- ----------
800       SMITH
950       JAMES
1100      ADAMS
1250      MARTIN ------>
1250      WARD -------->
1300      MILLER
1500      TURNER
1600      ALLEN
2450      CLARK
2850      BLAKE
2975      JONES
3000      FORD ------>
3000      SCOTT ----->
5000      KING
```

<u>OPERATORS</u>

Arithmetic Operators:

+ - * /

These are used to perform Arithmetic calculations on user's own data and table data.

DUAL table:

It is a system defined table which contains only one column to perform calculations on users own data.

Arithmetic Calculations On Users data:

Ex:
```
select 200+300 from dual;

500
```

Ex:
```
select (90000*10)/100  "10% of 90000" from dual;

10% of 90000
------------
9000
```

Ex:
```
select 2000+(0.10*5000)-300 " After calculation" from dual;
2200
```

Arithmetic Calculations On Table data:

Ex: Display emp salaries and 2% of salary as TA?

```
select sal " Basic Sal", (0.02*sal) " TA"  from emp;
```

Ex: Display employee salaries, 2% as TA, 5% as DA, 10% HRA, 4% as COMM and final salary?

```
select Sal " Basic", (0.02*Sal) " TA", (0.05*sal) "DA", (0.10*sal) "HRA",
        (0.04*sal) " Comm",
        (Sal + (0.02*Sal) + (0.05*sal) + (0.10*sal) + (0.04*sal) " Final Salary"
from emp;
```

RELATIONAL OPERATORS:-

These are used to compare values by specifying conditions on the columns.

< > = <= >=

WHERE clause:

In ***select*** query we can write conditions in this clause.

Syntax:
```
select cl1, cl2,......,cl-n / *
from table_name
where <conditions>
```

```
order by cl1, cl2,...,cln [asc/desc];
```

Ex: display salaries below 12000?

```
select sal " emp sal below 12000"  from emp_info where sal < 12000;
```

Ex: Display employee details who is getting above 12000 salary?

```
select * from emp_info where sal> 12000;
```

Ex: display the details of accounting dept?

```
select * from dept where dname='ACCOUNTING';
```

Ex: display the details of managers?

```
select * from emp where job='MANAGER';
```

Ex: display employee name and sal of empno 7788?

```
select ename,sal from emp where empno=7788;
```

Ex: Display employee details who joined before 1st jan 1981?

```
select * from emp where hiredate < '01-jan-81';
```

Ex:
```
select act_type,act_open_dt, count(actno) from cust_act_dtls
group by act_open_dt;
```

Ex:
```
select cityid,count(custid) " No. of accounts"
from cust_act_dtls
group by cityid;
```

ASSIGNMENTS:

Consider the below tables with estimated columns and then practise below questions.

CUST_DTLS
CUST_Act_DTLS
ACT_TYPES_INFO
PROD_DTLS
EMP
DEPT

1) Fetch all clerks information
2) Display all departments information located at CHICAGO?

3) Display product details manufactured in the current year only?
4) Get the details of cutomers accounts who opened the accounts before this year?
5) Get all SALARY account details?
6) Display customer names and mobile numbers from the city 'Texas'?

```
select cname,mobile from cust_dtls where city='Texas';
```

7) Get the information of Trading account?

8) Display only Expired product details?

```
select * from prod_dtls where exp<sysdate;
```

SPECIAL OPERATORS:

BETWEEN	It supports specific range of values.
IN	It supports specific list of values
IS NULL	It is used to check the column value is null or not, if it is null display output
LIKE	It is used to represents sequence of chars / specific strings

Syntax:-1 **BETWEEN**

```
select cl1, cl2,......,cl-n / *
from table_name
where <ColumnName> BETWEEN <start_value>  AND  <end_value>
order by cl1, cl2,...,cln [asc/desc];
```

Note: In the above syntax it includes both start value and end value.
BETWEEN Operator supports both Numeric range and Date range

Syntax: **IN**

```
select cl1, cl2,......,cl-n / *
from table_name
where <ColumnName> IN(val1, val2,val3,.....)
order by cl1, cl2,...,cln [asc/desc];
```

Note: IN operator works on Numeric, string and Date type data.

Syntax: **IS NULL**

```
select cl1, cl2,......,cl-n / *
from table_name
where <ColumnName> IS NULL
order by cl1, cl2,...,cln [asc/desc];
```

Note: It works on only null values and it is independent of data type of column.

Syntax: **LIKE**

```
select cl1, cl2,......,cl-n / *
from table_name
where <ColumnName> LIKE'string'
order by cl1, cl2,...,cln [asc/desc];
```

LIKE:

It uses 2 symbols

_ (underscore)	represents anyone char
%	represents any number of chars

Ex: display salaries between 2000 and 3000 in ascending order?

```
select sal from emp where sal between 2000 and 3000 order by sal;
```

Ex: display employee details who is joined in 1981?

```
select * from emp where hiredate between '01-jan-81' and '31-dec-81';
                              or
select * from emp where hiredate like'%81';
```

Ex: display emplyees working like clerks and managers?

```
select * from emp where job IN('CLERK','MANAGER');
```

Ex: display employee names and salaries who is getting any one of following salary?

1250,3000,5000

```
select ename,sal from emp where sal in(1250,3000,5000);
```

Ex: display employee id,name,sal,comm who is not getting comission?

```
select empno,ename,sal,comm from emp
where comm is null;
```

Ex: dispaly 3 digit salaries?

```
        select sal from emp where sal like'___';
```

Ex: display names of emps begins s?

```
        select ename from emp where ename like'S%';
```

Ex: display employees joined in 87?

```
        select * from emp where hiredate like'%87';
```

22-may-87
02-feb-87
11-oct-87

```
select * from emp;

select * from emp where sal between 1000 and 2000;

select * from emp where sal not between 1000 and 2000;

select * from emp where hiredate between '01-jan-81' and '31-dec-81' order by hiredate;

select * from emp where hiredate not between '01-jan-81' and '31-dec-81' order by hiredate;

select * from emp where job in('CLERK','SALESMAN');

select * from emp where job not in('CLERK','SALESMAN');

select * from emp where deptno in(10,20);

select * from emp where hiredate in('19-apr-87','23-jan-82') ;


create table sample as select * from emp;
select * from sample;

update sample set deptno=null where empno
in(7499,7566,7698,7788,7876,79007902,7934);

--Display 3-digit salaries
select sal from emp where sal like'___';


--Display salaries begining with digit "2"?
select sal from emp where sal like'2%';

--Display employee names begins with "J" and ends with "S"?
select ename from emp where ename like'J%S';

--Display 4-char length employee names?
select ename from emp where ename like'____';
```

--Display 4-char length employee names ends with "D"?
select ename from emp where ename like'___D';

--Display employee names,salaries, hiredates joined inn the year " 81"?
select ename,sal,hiredate from emp where hiredate like'%81';

RELATION NEGATION OPERATORS:

!= (or) <> (or) ^= (NOT EQUAL TO)

NOT BETWEEN

NOT LIKE

NOT IN

IS NOT NULL

Ex: Display all emps details except SALESMAN?

select * from emp where job<>'SALESMAN';

Ex: Display employee details not joined in the last year?

select * from emp where hiredate NOT BETWEEN '01-jan-14' and '31-dec-14';

ASSIGNMENTS:

1) Display customer account details whose balance is at least 10000 and at most 100000?

2) Display unknown account details?

3) Display customer details whose gender is unknown?

4) Display customers from the citites 'TEXAS ' and 'CHICAGO'?

5) Display Product details manufactured in january of this year?

6) Display product details whose warrenty is finished in the last year?

7) Display customer names having a char 'K'?

8) Display customer details who is living in 6 char length cities?

LOGICAL OPERATORS:

These are used to specify Multiple conditions in the where clause.

AND Display output if all conditions are true.
If any one condition was failed then it will not display output.

OR Display output if anyone condition is true.
If all conditions are false then it will not display output.

Syntax:

```
SELECT cl1,cl2,....., / *
FROM <table_name>
WHERE  <cond-1> [ AND / OR ] <cond-2> [ AND / OR ] <cond-3> [ AND / OR ].......
ORDER BY cl1, cl2,...... [ ASC /DESC];
```

Ex: Display manager details getting above 2500 sal?

```
select * from emp where job='MANAGER' and sal>2500;
```

Ex: Display clerks and salesman details if their salary at least 1000 and atmost 1500?

```
select ename,sal,job from emp
where job in('CLERK','SALESMAN') AND sal between 1000 and 1500;
```

Ex: Display salary account details having below 100000 balance?

```
select * from cust_act_dtls where act_type='sal' and bal <100000;
```

Assignments:

Ex: Display tablet or mobile information if their cost min 10000 and max 15000?

Ex: Display product details if they were manufactured in current year and min cost 2000 and max cost 10000?

```
                    CUST_ACT_DTLS
                    -------------
ACTNO     ACT_TYPE  ACT_OPEN_DT    ACT_BAL         CUST_ID
```

Ex: Display "male" customers from "texas" and "female" customers from "chicago"?

```
                    CUST_DTLS
                    ---------
     CUST_ID        CUST_NAME CUST_CITY   CUST_GENDER
     CUST_MOBILE
```

```
select * from cust_dtls
where (gender='male' and city='texas')
                or
      ( gender='female' and city='chicago');
```

EX: Display employee details joined in 87 year or working under deptno 10?

```
select * from emp where hiredate like'%87' or deptno=10;
```

Ex: Display trading account details having min balance 10000 and savings account details having min balance 100000?

DML COMMANDS

UPDATE:
It is used to update old values with new values within the table. By default it updates all values in the column

Updating single column value:

Syntax:
```
update <table_name>
set colname= value / expression  where <condition>;
```

Note: Without condition update command change all values in the column.

Updating multiple column values:

Syntax:
```
update <table_name>
set colname1= value / expression,
    colname2= value / expression,
    colname3= value / expression
          :
  where <condition>;
```

Ex: update the commission of 7369 as 500?

```
update emp set comm=500 where empno=7369;
```

Ex: update all emps commissions as 1000?

```
update emp set comm=1000;
```

Ex: update the salesman salary with 20% increment , change their designition as Sr.SALES who joined before 2005?

```
update emp
set sal=sal+(0.20*sal),
   job='Sr.SALES'
where job='SALESMAN'        AND  hiredate < '01-jan'05';
```

DELETE:

It is used to delete the records from the table. By default it deletes all the records.

Syntax: delete from <table_name> where <cond>;

Ex: delete all customer details?

```
delete from cust_dtls;
```

Ex: delete employees information who is not getting any commission?

```
delete from emp where comm is null;
```

Note:
We can get the back the deleted records / data within the current session by using *ROLLBACK.*

Examples:

```
select * from sample;

update sample set comm=500 where empno=7369;

update sample set comm=1000;

update sample set sal=sal+(0.20*sal),comm=500 where job='SALESMAN';

--Delete employee whose deptno is unknown?

delete from sample where deptno is null;

delete from sample;


insert into sample values(1234,'dinesh','developer',null,'12-may-10',5000,2300,40);
```

ROLLBACK;

DDL COMMANDS:-

CREATE **ALTER** **DROP** **TRUNCATE**

ALTER:

It is used to change the structure of the table by

i) adding new columns

syn: alter table <table_name> ADD <col_name> datatype(size);

syn:

FOR ADDING MULTIPLE COLUMNS

```
alter table <table_name> ADD
                        (
                        <col_name> datatype(size),
                        <col_name> datatype(size),
                                :
                                :
                        );
```

ii) deleting existed columns

syn: alter table <table_name> DROP COLUMN <col_name>;

iii) changing the datatype and size (increasing / decreasing) of column

syn: alter table <table_name> MODIFY <colname> new_datatype(new_size);

Note:

A) If the column is empty then we can do the following

--We can change from any data type to any data type.

--We can change any size to any size .

B) If column is not empty then,

--Number type and CHAR type column size can be decreased but can be increased.

--Data types can be changed from CHAR TO VARCHAR2 AND vice versa, NUMBER(p) to NUMBER(p,s) BUT NOT VICE VERSA.

iv) Rename a column with New Name.

syn: alter table <table_name>
RENAME COLUMN <oldcolname> **to** <newcolname>;

Ex: alter table emp rename column ename to empnames;

3) DROP:

It is used to delete any data base object.

How to delete the table?

Ex: drop table <table_name>;

4) **TRUNCATE**:
It deletes all the data from the table, it cannot deletes the partial data from a table
Here the deletion is permanent. We cannot get back or restore deleted data.

Syntax: truncate table <table_name>;

Ex: truncate table customers;

Ex: let us consider a table stud_dtls with columns rno ,sname, fee

stud_dtls
----

rno	sname	fee
number(2)	varchar2(5)	number(5)

Ex: change the above table to maintain course name for each student?

alter table stud_dtls add course_name varchar2(10);

Ex: change the above table to maintain gender and mobile number for each student?

alter table stud_dtls add (gender char, mobile number(10));

Ex: from the above table delete student mobile number column?

alter table stud_dtls drop column mobile;

Ex: increase the student name column size to 20?

alter table stud_dtls modify sname varchar2(20);

Ex: change the name of column sname to stud_name?

alter table stud_dtls RENAME sname to STUD_NAME;

Ex:

```
create table student
(
rno  number(2),
name varchar2(5),
fee   number(5)
);

insert into student values(1,'a','12000');
insert into student values(2,'x','10000');

alter table student add mobile number(10);

select * from student;

update student set mobile=9898989898 where rno=1;

alter table student add (gender  char, course varchar2(10));

select * from student;

alter table student drop column mobile;

select * from student;

update student set course='oracle';

select * from student;

insert into student values(3,'abc','11000','m','Unix &Scripting');

delete from student where course='Unixnull';

alter table student modify course  varchar2(20);

alter table student modify fee number(5);

alter table student rename column rno to rollno;
```

TCL (TRANSACTION CONTROL LANGUGE) COMMANDS:

Generally, DML operations on table data are considered as transactions.

1) COMMIT

It is used to make permanent the user transactions(DML operations) on the table.

Note: Once a transaction made permanent then we cannot cancel it

2) ROLLBACK:

It is used to cancel the user transaction.

3) SAVEPOINT

It is used to save a set of transactions under a name.

SAVEPOINT <name>;

Examples:

```
create table cust
(
cid  char(3),
cname  varchar2(20)
);

insert into cust values('c00','Sanju');
insert into cust values('c01','Manoj');

select * from cust;

rollback;

select * from cust;

insert into cust values('c00','Sanju');
insert into cust values('c01','Manoj');

commit;
select * from cust;

rollback;
select * from cust;
```

Ex-2:

```
Insert into cust values('c02','hellen','Female','xx','hyd');
delete from cust where cname='Sanju';
savepoint s1;
update cust set phone='aa' where cname='hellen';
savepoint s2;

delete from cust;

select * from cust;

rollback to s2;

rollback;
```

DCL commands: [Data Control Language commands]

By using these commands the DBA can assign permissions or get back permissions to / from the users on the data base objects.

GRANT
used to assign permissions.

Syntax:

GRANT <Privillage list > **On** <schemaName.UserName.objectName> **TO** <servername.schemaname.username>;

REVOKE:
Used to get back permissions

Syntax:
REVOKE <Previllage List> **On** <schemaName.UserName.objectName> **FROM** <servername.schemaname.username>;

PREVILLAGE LIST:

SELECT
INSERT
UPDATE
DELETE
ALTER

Note: All previllages are represented by one keyword "ALL"

Ex:

```
                    ORCL
                     |
              -------------------------
              |                      |
              USER_1                          USER_2
              |                      |
      ____________________           ____________________
      |              |               |              |
      Products       companies       customers      sales
```

Ex: Assign SELECT previllage to the user-2 on the table PRODUCTS created by USER-1

GRANT SELECT ON ORCL.USER_1.PRODUCTS TO USER_2;

Ex: How do i cancel the above permission?

REVOKE SELECT ON ORCL.USER_1.PRODUCTS FROM USER-2;

Ex: How to assign CREATE VIEW permission to user-2?

```
GRANT CREATE VIEW  To user_2;
```

Ex: How do i cancel all previllages on all objects to the user-1?

```
REVOKE ALL from  USER_1;
```

Data Integrity Constraints

CONSTRAINTS:

Constraints are set of rules / business rules which will be defined at DDL level.
Constraints enforce the data base to allow only valid values in to the tables.
Constraints ensure the user to fetch only valid / complete and accurate data from the database.

Categories of Constraints: 3

1) **Key** Constraints
2) **Domain** Constraints
3) **Referential Integrity** constraints

1) KEY CONSTRAINTS:

These constraints check the individual values in to a column according to Business.
These are divided into 3 types

a) **UNIQUE**
It doesn't allow duplicates but allows null values.

Ex: This constraint is suitable for maintaining phone numbers, mail id, etc...

b) **NOT NULL**
It doesn't allow null values but allows duplicates.

Ex: Emp Names, Cust Names, ...

c) **PRIMARY KEY**
It doesnt allow duplicates and null values. Generally a Primary key is used to identify any record in a table **uniquely**. *"Only one primary key is allowed per a table"*.

Primary key is of 2 types.

Simple *primary key*
If a Primary key Defined on a single column then it is known as Simple Primary key.

Composite *primary key*
If a Primary key constraint defined on more than one column then it is known as Composite Primary Key.

(***Max numbers of columns in to a composite Primary key are 32 columns)***

Composite Primary Key: primary key (custid,prodid,timeid)

SALES_DTLS
==========

custid	prodid	timeid	qty	sales_amount
c1	p3	jan5-15	100	100000
c1	p2	jan5-15	100	200000
c2	p3	jan5-15	100	100000
c1	p3	jan7-15	100	150000
c5	p1	jan5-15	200	300000

Syntax: Creating a table with **key** constraints:

```
Create table  <table name>
(
col1  data type(size) <constraint_name>,
col2  data type(size) <constraint_name>,
:         :         :         :

);
```

Ex: create a table student with columns rno, sname, course, fee and mobile along with constraints pk, nn, nn, nn and unique respectively?

```
Create table student
(
rno      number(2) primary key,
sname varchar2(10) not null,
course varchar2(15) not null,
fee      number(5) not null,
mobile char(10) unique
);
```

```
insert into student values(1,'a','oracle',9000,'8989898989');
insert into student values(0,'b','java',2000,'8787878787');
insert into student values(2,'x','oracle',9000,null);
insert into student values(11,'s','abc',100,null);
```

data:-

```
 RNO          SNAME        COURSE       FEE          MOBILE
----------    ----------   ---------------  ----------   ----------
      1       a            oracle        9000        8989898989
      0       b            java         2000         8787878787
      2       x            oracle       9000
     11       s            abc          100
```

ERROR GENERATING RECORDS:

```
insert into student values(1,'kiran','java',2300,null);
insert into student values(null,'kiran','java',2300,null);
insert into student values(12,null,'java',2300,null);
insert into student values(1,'kiran',null,2300,null);
insert into student values(1,'kiran','java',null,null);
insert into student values(1,'kiran','java',2300,8989898989);
```

Note: Even after the key constraints on the table, still we have invalid values.
We can eliminate them by using **DOMAIN** constraints.

Displaying constraints information on a table:-

In oracle database, all constraints stored in a system defined table called ***USER_CONSTRAINTS***.
Each constraint named and numbered uniquely like, ***SYS_Cn*** (System defined Constraint).

To Fetch data from this table use the below example.

Ex:
```
select constraint_name,constraint_type from USER_CONSTRAINTS where
table_name='STUDENT';
```

```
CONSTRAINT_NAME              CONSTRAINT_TYPE
------------------------------ ---------------
SYS_C007050               C --either Not null or Check
SYS_C007051               C
SYS_C007052               C
SYS_C007053               P --Primary key
SYS_C007054               U --Unique key
SYS_C007055               R –Foreign key
```

CONSTRAINTS with user-defined names:

SYN: <col> datatype(size) Constraint <User defined name><actual constraint name>,
<col> datatype(size) Constraint <User defined name><actual constraint name>,

```
Ex:     <col>  datatype(size) Constraint  pk_rno_student Primary key,
```

```
create table stud_dtls
        (
        rno     number(2)  constraint  pk_rno_stud  primary key,
        sname varchar2(20) constraint  nn_sname_stud not null,
        course varchar2(7) constraint  nn_course_stud  not  null,
        fee     number(5) constraint nn_fee_stud not null,
        mobile number(10) constraint  uk_mobile_stud  unique
        );
```

```
create table student_dtls
        (
        rno     number(2)  constraint  pk_rno_student  primary key,
        constraint  ck_rno_student  check (rno between 1 and 60),
        sname varchar2(20) constraint  nn_sname_student not null,
        course varchar2(7) constraint  nn_course_student  not  null,
        constraint  ck_course_student check (course In('cse','ece','eee','it')),
        fee     number(5) constraint nn_fee_student not null,
        constraint  ck_fee_student  check (fee between  30000 and 40000),
        mobile number(10) constraint  uk_mobile_student  unique
        );
```

```
select constraint_name,constraint_type from user_constraints where
table_name='STUDENT_DTLS';

CONSTRAINT_NAME               CONSTRAINT_TYPE
------------------------------     ---------------
NN_SNAME_STUDENT                    C
NN_COURSE_STUDENT                   C
NN_FEE_STUDENT                      C
CK_RNO_STUDENT                      C
CK_COURSE_STUDENT                   C
CK_FEE_STUDENT                      C
PK_RNO_STUDENT                      P
UK_MOBILE_STUDENT                   U

 8 rows selected
```

DOMAIN constraints:

It is used to define a valid range / valid list of values on a column by using the keyword *CHECK*.

CHECK uses 2 operators.

```
BETWEEN      to define range
IN           to define list of values.
```

Syntax:

```
create table  <table_name>
        (
        col1  datatype(size) <constraint_name>,
        col2  datatype(size) <constraint_name>,
        :         :         :         :,

        CHECK (col1 BETWEEN begin_value AND end_value),
        CHECK (col2 IN (val1, val2,........,)),
        ....
        ....

        );
```

Ex: create the above table along with below domain constraints:

```
--rno should be between 1 and 60
--course names are oracle, sql server and unix
--Min fee is 5000 and max fee  10000
```

```
create table stud_dtls
 (
 rno number(2) constraint pk_rno_stud primary key,
 sname varchar2(10) constraint nn_name_stud not null,
 course varchar2(15) constraint nn_course_stud not null,
 fee number(5) constraint nn_fee_stud not null,
 mobile char(10) constraint uk_mobile_stud unique,
 constraint ck_rno_stud check (rno between 1 and 60),
 constraint  ck_course_stud check (course in('oracle','sql server','unix')),
 constraint ck_fee_stud check (fee between 5000 and 10000)
 );
```

Records:

```
insert into stud_dtls values(1,'a','oracle',7000,1212);
 insert into stud_dtls values(0,'b','sql server',7000,null);
 insert into stud_dtls values(61,'a','oracle',7000,null);
 insert into stud_dtls values(12,'b','sql server',7000,null);
 insert into stud_dtls values(11,'ajay','unics',10000,2212);
 insert into stud_dtls values(21,'hari','unix',17000,1211);
insert into stud_dtls values(21,'hari','unix',11000,1211);
```

```
       RNO SNAME      COURSE            FEE MOBILE
---------- ---------- --------------- ---------- ----------
         1 a          oracle           7000 1212
        12 b          sql server       7000
        21 hari       unix            11000 1211
```

CREATING A TABLE WITH USER-FRIENDLY NAMES TO THE CONSTRAINTS:-

```
create table s_dtls
 (
 rno number(2)                    CONSTRAINT PK_RNO_S_DTLS    primary key,
 sname varchar2(10)  CONSTRAINT NN_SNAME_S_DTLS          not null,
 course varchar2(15)  CONSTRAINT NN_COURSE_S_DTLS       not null,
 fee number(5)                    CONSTRAINT NN_FEE_S_DTLS     not null,
 mobile char(10)         CONSTRAINT UK_MOB_S_DTLS    unique,
 CONSTRAINT CK_RNO_S_DTLS      check (rno between 1 and 60),
 CONSTRAINT CK_COURSE_S_DTLS   check (course in('oracle','sql server','unix')),
 CONSTRAINT CK_FEE_S_DTLS    check (fee between 10000 and 20000)
 );
```

How do i display constraints information of a table?

```
SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE FROM USER_CONSTRAINTS
WHERE TABLE_NAME='s_dtls'; ----> this won't work
SELECT CONSTRAINT_NAME,CONSTRAINT_TYPE FROM USER_CONSTRAINTS
WHERE TABLE_NAME='S_DTLS';
```

```
CONSTRAINT_NAME              CONSTRAINT_TYPE
------------------------------ ---------------
NN_SNAME_S_DTLS              C
NN_COURSE_S_DTLS             C
NN_FEE_S_DTLS                C
CK_RNO_S_DTLS                C
CK_COURSE_S_DTLS             C
CK_FEE_S_DTLS                C
PK_RNO_S_DTLS                P
UK_MOB_S_DTLS                U
```

constraint_type	Meaning
C	check or not null
P	Primary Key
U	Unique key
R	Foreign key

Assignment:

i) create customers table with columns custid,custname,city,gender,mailid,phone, Address with the constraints   Pk,NN,NN,NN, and Unique and Unique Respectively.

NORMALIZATION AND DENORMALIZATION CONCEPTS:

Consider the below tables and data.

DEPT

DNO	**DNAME**	**LOC**
10	Production	Hyderabad
20	Sales	Hyderabad
30	Finance	Chennai

EMP

Eid	**Ename**	**Salary**
1	A	2000
2	X	1200
3	A	3400
4	Z	5000
5	C	1000
6	S	1300
7	D	2300
8	X	1200
9	B	2200

Note:
By using above tables we are unable fetch the complete data of an object, like, department name of any employee, number of employees in dept and etc.

The solution for such kind of requirements we have 2 methods.

1) Maintaining all the information in one big table [*DENORMALIZED DATA*]

2) Maintaining data in different tables and implement Physical relationships between the tables [*NORMALIZED DATA*]

1) DENORMALIZATION

Maintaining all information in one big table is known as De-normalized method.

emp_dept_details

Eid	Ename	Salary	Dno	Dname	Loc
1	A	2000	10	Production	Hyderabad
2	X	1200	10	Production	Hyderabad
3	A	3400	10	Production	Hyderabad
4	Z	5000	10	Production	Hyderabad
5	C	1000	20	Sales	Hyderabad
6	S	1300	20	Sales	Hyderabad
7	D	2300	20	Sales	Hyderabad
8	X	1200	30	Finance	Chennai
9	B	2200	30	Finance	Chennai

Note:
From the above table we will get the required information, but it has

- data duplicacy
- it occupies more disk space
- data retrieval time is very high.

Disk space: 6X9=54 kb

2) NORMALIZATION:

The Process of dividing the above big table in to sub tables until the data duplicacy is maximum reduced is called normalization process.

i) *Ist NF(normal form):*

Dividing the table into sub tables based on repeated groups of data.

	emp ---		dept ------------		
eid ---	ename -----	sal ----	dno ---	dname ------	loc ----
1	a	2000	10	production	hyderabad
2	x	1200	10	production	hyderabad
3	a	3400	10	production	hyderabad
4	z	5000	10	production	hyderabad
5	c	1000	20	sales	hyderabad
6	s	1300	20	sales	hyderabad
7	d	2300	20	sales	hyderabad
8	x	1200	30	fin	chennai
9	b	2200	30	fin	chennai

Ex:

Emp Table:

```
create table emp
as
select eid,ename,sal from emp_dept_details;
```

DEPT table:

```
create table dept
as
select dno,dname,loc from emp_dept_details;
```

ii) IInd NF:
Eleminating duplicates and defining primary keys

```
        emp                     dept
        ---                     ------------
eid     ename sal               dno           dname         loc
---     ----- ----              ---           ------        ----
1       a     2000              10            production    hyderabad
2       x     1200              20            sales         hyderabad

3       a     3400              30            fin           chennai

4       z     5000              PK
5       c     1000
6       s     1300
7       d     2300
8       x     1200
9       b     2200
PK
```

Ex: Eleminating Duplicate records

```
create table dept1
as
select distinct dno,dname,loc from dept;
```

iii) IIIrd NF:

Implementing relationships between the tables by using Primary key of dept table , define the foreign key column under emp table.

```
        emp                         dept
        ---                         ----
```

eid	ename	sal	emp_dno	dno	dname	loc
---	-----	----	-------	---	------	----
1	a	2000	10	10	production	hyderabad
2	x	1200	10	20	sales	hyderabad
3	a	3400	10	30	fin	chennai
4	z	5000	10	PK		
5	c	1000	20			
6	s	1300	20			
7	d	2300	20			
8	x	1200	30			
9	b	2200	30			
PK			FK			

It occupies less disk space and max .data duplicacy is reduced.

```
disk space:  4X9=36
             3X3=9

        45KB
```

Advantages:
--Searching for required data is as much as fast
--And data retrieval is fast
--Max data duplicacy is eleminated.
--Occupy less Disk space.

<u>3) REFERENTIAL INTEGRITY CONSTRAINTS</u>:

Used to implement PHYSICAL relationship between the tables by using primary key of one table and we can define foriegn key in other table.

Foriegn key column contains only values from primary key. Foriegn key contains duplicates and null values also.

--A table which contains primary key is considered as parent /Master/Base table.
--A table which contains foriegn key is known as child table /Detailed table/ Derived table.

REFERENCES:
we can use this keyword in the "creation of child table and to define foriegn key column".

Ex: create comp_dtls as parent table

Ex: create prod_dtls as child table

```
        create table  comp_dtls
        (
        cmpId  char(5) constraint pk_cmpid_comp_dtls primary key,
        cmpName       varchar2(20) not null,
        cmpCountry    varchar2(20) not null,
        constraint ck_country_cmp
         check (cmpcountry IN('india','usa','japan','uk'))
                );

        insert into comp_dtls values('cmp01','sony','japan');
        insert into comp_dtls values('cmp02','wipro','india') ;
        insert into comp_dtls values('cmp03','Philips','india');
        insert into comp_dtls values('cmp04','semantic','usa');


        create table prod_DTLS
        (
        pid       char(4) primary key,
        pname varchar2(20) not null,
        cost      number(7,2),
        mfg       date,
        warrenty  varchar2(10),
        cmpId  char(5),
        constraint fk_prod_cmpid  FOREIGN KEY(cmpId) REFERENCES comp_dtls(cmpId)
/* foriegn key column */
        );


        insert into prod_dtls values
('p001','smart phone',34000,'12-may-14','1 year','cmp01');
        insert into prod_dtls values
('p002','laptop',54000,'03-feb-14','3 years','cmp01');
        insert into prod_dtls values
('p003','Television',24000,'08-aug-14','5 years','cmp03');
        insert into prod_dtls values
('p004','Home Theatre',55000,'11-aug-13','2 years','cmp03');
        insert into prod_dtls values
('p005','Mobile',24000,'08-aug-14','1 year',null);
        insert into prod_dtls values
('p006','vmware',64000,'22-oct-10','1 year','cmp04');
```

Ex: create the following tables and Implement relationships accordingly?

1) Cust_dtls (parent table)

```
        cno      cname city      gender mobile
        |
        Primary key
```

2) Act_types (parent table)

Act_type	act_name	desc
SB	Savings Bank	
DEMAT	Trading account	
CA	Current account	
Primary key		

3) cust_act_dtls (Child table)

Actno	Act_type	Act_open_date	Act_bal	cno
PK	Foreign key			Foreign key

sample examples:

```
create table student
        (
        rno     number(2) primary key,
        sname varchar2(10) not null,
        course varchar2(15) not null,
        fee     number(5) not null,
        mobile char(10) unique
        );
 insert into student values(1,'a','oracle',9000,1212);
 insert into student values(11,'x','oracle',6000,1213);
 insert into student values(0,null,'oracle',9000,1214);
 insert into student values(0,'B','oracle',9000,1214);
 insert into student values(10,'Ajay','orcl',9000,1215);
 insert into student values(20,'A','xyz',99000,null);
 insert into student values(13,'A','unix',19000,null);
 insert into student values(23,'BAC','unix',0,null);

 select * from student;


 create table stud
        (
        rno     number(3) constraint pk_rno_stud primary key,
        sname varchar2(10) constraint nn_sname_stud not null,
        course varchar2(15) constraint nn_course_stud not null,
        fee     number(5)  constraint nn_fee_stud not null,
        mobile char(10) constraint uk_mobile_stud unique,
 constraint ck_rno_stud CHECK ( rno between 1 and 100),
 constraint ck_course_stud CHECK( course in('oracle','unix')),
 constraint  ck_fee_stud CHECK(fee between 5000 and 20000)
 );

        );
 insert into stud values(1,'a','oracle',9000,1212);
 insert into stud values(11,'x','oracle',6000,1213);
 insert into stud values(10,'C','oracle',9000,1214);
 insert into stud values(20,'B','oracle',9000,1211);
 insert into stud values(12,'Ajay','oracle',9000,1215);
 insert into stud values(22,'A','unix',20000,null);
 insert into stud values(13,'A','unix',19000,null);
 insert into stud values(23,'BAC','unix',6000,null);

 select * from stud;

 alter table stud drop column course;
 alter table stud drop column fee;

 select * from stud;

 create table course
 (
 cid varchar2(10) constraint pk_course_id  primary key,
 Name  varchar2(20) constraint nn_name_course not null,
 fee        number (5) constraint nn_fee_course  not null
 );

 insert into course values('C1','Oracle',13000);
 insert into course values('C2','Unix',10000);
 insert into course values('C3','Linux',10000);
 insert into course values('C4','Sql Server',10000);

 select * from course;

 select * from stud;

 alter table stud add course_id varchar2(20);

 alter table stud add constraint fk_course_id_stud FOREIGN KEY(course_id)
 REFERENCES course(cid);

 update stud set course_id='C1'
 where rno in(1,10,20,22,23);

 update stud set course_id='C2' where course_id is null;
 select * from stud;

 select s.sname,s.mobile,c.name,c.fee from stud s , course c
 where s.course_id=c.cid;
```

MERGE statement:

Oracle’s MERGE statement is used for situations when you want to do an "upsert" i.e. update existing rows in a table or insert new rows depending on a match condition. This is typically the case when you have to synchronize a table periodically with data from another source (table/view/query).

Syntax:

```
MERGE into <target table>
USING
<source table/view/result of subquery>
ON
<match condition>
WHEN MATCHED THEN
<update clause>
<delete clause>
WHEN NOT MATCHED THEN
<insert clause>
```

Example:

```
SQL> select * from student;

      ID NAME              SCORE
---------- --------------- ----------
       1 Jack              540
       2 Rose
       3 William            650
       4 Caledon             620
       5 Fabrizio           600
       6 Thomas
       7 Ruth              680
       8 Spacer             555

8 rows selected.

SQL> select * from student_n;

      ID NAME              SCORE
---------- --------------- ----------
       7 Ruth              690
       8 Spicer            620
       9 Wallace            600
      10 Lizzy
      11 Brock              705
```

As you can see, the following actions are required on table STUDENT:

1 row for id#7 to be corrected for score: Ruth had scored 690, not 680.
1 row for id#8 to be corrected for name: the student is called Spicer, not Spacer.
3 new rows (ids#9,10,11) to be inserted into STUDENT table.

Note: 5 rows should get processed in all.

```
 merge into student a
  using
```

```
   (select id, name, score
    from student_n) b
 on (a.id = b.id)
 when matched then
   update set a.name = b.name
        , a.score = b.score
 when not matched then
   insert (a.id, a.name, a.score)
   values (b.id, b.name, b.score);

5 rows merged.
```

Let’s check the values in table STUDENT now.

```
SQL> select * from student;

        ID NAME                 SCORE
---------- --------------- ----------
         1 Jack                   540
         2 Rose
         3 William                650
         4 Caledon                620
         5 Fabrizio               600
         6 Thomas
         7 Ruth                   690
        11 Brock                  705
        10 Lizzy
         9 Wallace                600
         8 Spicer                 620

11 rows selected.
```

Sure enough, 5 rows have got merged as expected – 2 updates + 3 inserts.

SET OPERATORS

These Operators are used to select same data type of data and Equal number of column's data from multiple tables.

Or

These operators will display combined data from multiple tables.

1) UNION

It will display combined data from multiple tables without duplicates

2) UNION ALL

It will display combined data from multiple tables with duplicates

3) INTERSECT

It will display common data from multiple tables (From multiple Select stmts)

4) MINUS

It will display values from first selection by eliminating values which are repeating in second selection

For example:

s1={a,b,c,d}   s2={x,y,z,c,b}

```
1) select * from s1
        union
        select * from s2;

output: {a,b,c,d,x,y,z}

2) select * from s1
        union all
        select * from s2;

output: {a,b,c,d,x,y,z,c,b}


3)select * from s1
        intersect
        select * from s2;

output: {c,b}

4)select * from s1
        minus
        select * from s2;

output: {a,d}
```

Sample Tables:

```
CREATE TABLE CUST_BR1
(
CID CHAR(3),
CNAME VARCHAR2(20),
MOBILE NUMBER(10),
CITY  VARCHAR2(20),
GENDER  VARCHAR2(10)
);

INSERT INTO CUST_BR1 VALUES('C1','VIJAY',1212121212,'HYD','MALE');
INSERT INTO CUST_BR1 VALUES('C2','JOHN',1313131313,'DELHI','MALE');
INSERT INTO CUST_BR1 VALUES('C3','SWATHI',1414141414,'HYD','FEMALE');
```

```
CREATE TABLE CUST_BR2
(
CID CHAR(3),
CNAME VARCHAR2(20),
MOBILE NUMBER(10),
CITY  VARCHAR2(20),
GENDER  VARCHAR2(10)
);

INSERT INTO CUST_BR2 VALUES('C1','KIRAN',9898989898,'HYD','MALE');
INSERT INTO CUST_BR2 VALUES('C2','JOHN',1313131313,'DELHI','MALE');
INSERT INTO CUST_BR2 VALUES('C3','LAKSHMI',8989898989,'DELHI','FEMALE');

CREATE TABLE CUST_BR3
(
CID CHAR(3),
CNAME VARCHAR2(20),
MOBILE NUMBER(10),
CITY  VARCHAR2(20),
GENDER  VARCHAR2(10)
);

INSERT INTO CUST_BR3 VALUES('C1','KIRAN',9898989898,'HYD','MALE');
INSERT INTO CUST_BR3 VALUES('C2','JOHN',1313131313,'DELHI','MALE');
INSERT INTO CUST_BR3 VALUES('C5','VINAY',7878787878,'DELHI','MALE');
```

Examples:

```
--Display all customers info from all branches

select * from cust_br1
union all
select * from cust_br2
union all
select * from cust_br3;

--Get the customer details without duplicates

select * from cust_br1
union
select * from cust_br2
union
select * from cust_br3;

--DISPLAY COMMON CUSTOMER NAMES AND MOBILE NUMBERS FROM ALL
BRANCHES


SELECT CNAME,MOBILE FROM CUST_BR1
INTERSECT
SELECT CNAME,MOBILE FROM CUST_BR2
INTERSECT
SELECT CNAME,MOBILE FROM CUST_BR3;
```

```
--DISPLAY CUSTOMERS DETAILS WHO IS THE ONLY CUSTOMER FOR BRANCH 2

SELECT * FROM CUST_BR2
MINUS
(
SELECT * FROM CUST_BR1
UNION ALL
SELECT * FROM CUST_BR3
);
```

Note:
What are the limitations of set operators?

We need to select equal number of columns from each table
We need to select same data type of data in the same sequence under each select query.

Ex:

```
 select cname,mobile from cust_br1
 union all
 select cid from cust_br2;
select cname,mobile from cust_br1
*
ERROR at line 1:
ORA-01789: query block has incorrect number of result columns


SQL> select cname,mobile from cust_br1
  2  union all
  3  select mobile,cname from cust_br2;
select cname,mobile from cust_br1
       *
ERROR at line 1:
ORA-01790: expression must have same datatype as corresponding expression
```

<u>JOINS</u>

Joins are used to display multiple data types of data from multiple tables.

Types of joins: 4

i) CROSS JOIN

ii) EQUI JOIN / INNER JOIN

iii) SELF JOIN

iv) OUTER JOINS

<u>CROSS JOIN:</u>

It will display combination of data from multiple tables.
In this join, each value in the first table is mapped with all values in the second table. It will display all possible combinations of data from multiple tables.

Example:
consider
s1={a,b,c,d}   s2={d1,d2}

s1Xs2={(a,d1),(a,d2),(b,d1),(b,d2),.......}

```
syn:    select col1, col2,...,coln
        from table1, table2,....
        where <cond>
        order by col1, col2,....[ desc ];
```

Ex: Display employee names , salaries and their department names?

```
select ename,sal,dname from emp,dept;
```

		emp				dept		
eid	ename	sal	job	hiredate	deptno	deptno	dname	loc
1	a	1200	dev	11-oct-12	10	10	development	hyd
2	b	1100	dev	12-oct-12	10	20	testing	hyd
3	x	1000	admin	22-nov-12	30	30	finance	chennai
4	y	2300	dev	22-nov-12	10			

output:

```
a       1200    development-----> valid
a       1200    testing
a       1200    finance
b       1100    development-----> valid
b       1100    testing
b       1100    finance
x       1000    development
x       1000    testing
x       1000    finance---------> valid
```

```
y        2300   development-----> valid
y        2300   testing
y        2300   finance
```

Note:
In the above output marked combinations are valid and remaining are invalid according to physical table data.
So, Then in which case cross join will display only valid combination?

Ex: for all managers display 13% increment details?

```
select ename,sal,job,per_id,per_incr,desc
from emp,Percent_incr_dtls
where job='manager' and per_incr='13%';
```

Cross join examples:

```
CREATE TABLE "PER_INCR"
  (     "INCRID" NUMBER,
        "INCRVAL" VARCHAR2(20 BYTE),
        "DESCRIPTION" VARCHAR2(40 BYTE) DEFAULT null
  );
```

```
insert into per_incr values
(101,'5%','Min INcrement');
insert into per_incr values
(102,'10%','Second level INcrement');
insert into per_incr values
(103,'15%','3rd level INcrement');
insert into per_incr values
(104,'25%','Max level INcrement');
```

```
select * from emp;
select * from per_incr;
```

```
 INCRID      INCRVAL            DESCRIPTION
----------   -------------------- ----------------------------------------
       101   5%                  Min INcrement
       102   10%                  Second level INcrement
       103   15%                  3rd level INcrement
       104   25%                  Max level INcrement
```

Ex: display managers details with 15% increment?

```
select e.ename,e.sal,e.job,i.incrid,i.incrval,i.description
from emp e,per_incr i
where e.job='MANAGER' AND i.incrval='15%';
```

Ex:	Apply 5% for all clerks

```
select e.*, i.* from emp e, per_incr i
where e.job='CLERK' and i.incrval='5%';
```

EQUI JOIN / INNER JOIN:

A cross join is known as equi join if we specify join condition using '=' operator.
It will display only matched data from all tables.
A condition is known as join condition if it is specified between primary key of one table and foriegn key of other table.

```
syn:	select col1, col2,...,coln / *
	from table1, table2,....
	where  table1.pk=table2.fk and table2.pk=table3.fk.......;
```

Ex:	display employee names , salaries and corresponding department details?

```
	select ename, sal,dept.deptno,dname,loc
	from emp,dept
	where emp.deptno=dept.deptno;
```

output:
```
	a	1200	10		development	hyd
	b	1100	10		development	hyd
	x	1000	30		finance		chennai
	y	2300	10		development	hyd
```

Ex: Display employee details and dept details ?

```
select e.ename,e.sal,e.job,e.deptno,d.deptno,d.dname,d.loc from emp e, dept d
where e.deptno=d.deptno;
```

--display employee details and dept details for all managers?

```
select e.*,d.* from emp e, dept d
where e.job='MANAGER' and e.DEPTNO = d.DEPTNO;
```

--Display all clerks details and their department names?

```
SELECT E.*,D.DEPTNO,D.DNAME FROM EMP E, DEPT D
WHERE E.JOB='CLERK' AND E.DEPTNO=D.DEPTNO
ORDER BY SAL DESC;
```

Inner join

```
select ename, sal,deptno,dname,loc
from emp Inner join dept
ON emp.deptno=dept.deptno;
```

output:

```
    a       1200   10          development   hyd
    b       1100   10          development   hyd
    x       1000   30          finance       chennai
    y       2300   10          development   hyd
```

Ex: display customer name and city, customer actno,acttype and bal, act_name for all customers?

EQUI JOIN:

```
select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal,at.act_name
from cust_dtls cd, cust_act_dtls cad, act_types_info at
where cd.cno=cad.cust_code
        and
        cad.act_type=at.act_type;
```

INNER JOIN:

```
select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal,at.act_name
from cust_dtls cd INNER JOIN cust_act_dtls cad
ON cd.cno=cad.cust_code
        INNER JOIN   act_types_info  at
ON      cad.act_type=at.act_type;
```

Ex: Applying equi join on 5 tables?

```
   select t1.*,t2.*...
from t1,t2,t3,t4,t5
   where
   t1.col1=t2.col4
   and
   t2.col3=t3.col1
   and
   t3.col2=t4.col31
   and
   t4.col31=t5.col30;
```

Ex: Applying Inner join on 5 tables?

```
   select t1.*,t2.*...
from t1 Inner Join t2
```

```
ON
t1.col1=t2.col4       Inner Join t3


ON
t2.col3=t3.col1       Inner Join t4


ON
t3.col2=t4.col31      Inner Join           t5

ON
t4.col31=t5.col30;
```

Ex:	Display manager details and corresponding department details?

```
select e.*,d.*
from emp e, dept d
where  job='MANAGER'
                AND
        e.deptno=d.deptno;
```

or

```
select e.*,d.* from emp e inner join dept d
on e.deptno=d.deptno
where e.job='MANAGER';
```

Ex:

```
select e.eid,e.sal,e.job,d.dname
from emp,dept                           [ invalid query ]
where e.deptno=d.deptno;
```

NOTE:
In the above query the e and d are temporary alias names for the table emp and dept respectively. And these alias names are valid for current query only.

Ex:
--DISPLAY EMPLOYEE DETAILS AND DEPARTMENT DETAILS WHO IS WORKING UNDER accounting and sales?

```
SELECT E.*,D.* FROM EMP E INNER JOIN DEPT D
ON d.dname IN('ACCOUNTING','SALES')
  AND
E.DEPTNO = D.DEPTNO;
```

Ex:	display product and corresponding company details?

Ex:	display customer details and corresponding account details?
Ex:	display company name and its products details if the products are from the company "sony"?
Ex:	display expired product details and their corresponding company details?

EQUI JOIN / INNER JOIN EXAMPLES:

```
--comp_dtls--[comp_id, comp_name, country ]
              PK

--prod_dtls--[pid, pname, cost, mfg, warrenty, comp_id ]
              PK                           FK

--cust_dtls--[cid, cname, city, gender, mobile ]
              PK

--Time_dtls--[tid, tperiod, time_desc ]
              PK

--sales_dtls-[ cid, pid, tid, quantity, sale_amount]
              FK  FK   FK
       <----PK-------->
```

Ex: Display customer names, mobile, product names, cost, warrenty and company names?

```
select c.cname, c.mobile, p.pname,p.cost , p.warrenty, cmp.comp_name
from cust_dtls c, prod_dtls p, comp_dtls cmp, sales_dtls s
where
c.cid=s.cid
and
s.pid=p.pid
and
p.comp_id=cmp.comp_id;
```

Ex: Display product details along with its sales informations from the year 2014?

```
select p.*,s.* from prod_dtls p, sales_dtls s
where  s.tid IN('Q1-2014','Q2-2014','Q3-2014','Q4-2014')
       and
       s.pid=p.pid;
```

TABLES:

```
cust_dtls--[cid, cname, city, gender, mobile ]
            PK

act_types_dtls--[ act_type, act_name, act_description ]
                  PK

cust_act_dtls--[ actno, act_type, act_open_date, act_bal, cid]
```

PK   FK   FK

Ex: Display customer details and their account details who is from the city 'CHICAGO'?

Ex: Display above information if the customer held DEMAT account and with min balance 100000?

```
        select c.*,a.* from cust_dtls c, cust_act_dtls a
        where (c.city='CHICAGO'
                and
                (a.act_type='DEMAT' and a.act_bal>=100000))
                and
                c.cid=a.cid;
```

Practical Examples:

Ex: Write a query to display customer names, account names and respective Act balances?

By Using Equi Join:

```
select cd.cname,at.act_name,cad.act_bal
from CUST_DTLS cd,ACT_TYPES at,CUST_ACT_DTLS cad
where cd.cno=cad.cust_code  AND cad.act_type=at.act_type;
```

By using Inner Join:

```
select cd.cname,at.act_name,cad.act_bal
from CUST_DTLS cd Inner Join CUST_ACT_DTLS cad
ON cd.cno=cad.cust_code   Inner Join act_types at
ON cad.act_type=at.act_type;
```

```
SQL> select cd.cname,cad.actno
        from cust_dtls cd inner join cust_act_dtls cad
        on cd.cno=cad.cust_code;

CNAME       ACTNO
------------ -----------
Anil        20035201471
Kiran        20035201473
vinod        20035201472
vinod        20035201470
Madhu        20035201474
Rocky        20035201475
Ching Fu     20035201476

7 rows selected.

SQL> select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal from cust_dtls c
d inner join cust_act_dtls cad
  2  select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal from cust_dtls c
```

```
d inner join cust_act_dtls cad
  3  on cd.cno=cad.cust_code;
select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal from cust_dtls cd inn
er join cust_act_dtls cad
*
ERROR at line 2:
ORA-00905: missing keyword


SQL> select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal
  2  from cust_dtls cd inner join cust_act_dtls cad
  3  on cd.cno=cad.cust_code;

CNAME        CITY       ACTNO       ACT_T    ACT_BAL
------------ ---------- ----------- ----- ----------
Anil         Texas      20035201471 SAL        32000
Kiran        Chicago    20035201473 SB         23000
vinod        Delhi      20035201472 DEMAT     123000
vinod        Delhi      20035201470 SB         49000
Madhu        Delhi      20035201474 SAL        11000
Rocky        Texas      20035201475 SB         13000
Ching Fu     Chicago    20035201476 SAL        23000

7 rows selected.

SQL> select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal
  2  from cust_dtls cd inner join cust_act_dtls cad
  3  on cd.cno=cad.cust_code
  4  order by cad.act_bal;

CNAME        CITY       ACTNO       ACT_T    ACT_BAL
------------ ---------- ----------- ----- ----------
Madhu        Delhi      20035201474 SAL        11000
Rocky        Texas      20035201475 SB         13000
Kiran        Chicago    20035201473 SB         23000
Ching Fu     Chicago    20035201476 SAL        23000
Anil         Texas      20035201471 SAL        32000
vinod        Delhi      20035201470 SB         49000
vinod        Delhi      20035201472 DEMAT     123000

7 rows selected.

SQL> select cd.cname,cd.city,cad.actno,cad.act_type,cad.act_bal
  2  from cust_dtls cd inner join cust_act_dtls cad
  3  on cd.cno=cad.cust_code
  4  order by cad.act_bal desc;

CNAME        CITY       ACTNO       ACT_T    ACT_BAL
------------ ---------- ----------- ----- ----------
vinod        Delhi      20035201472 DEMAT     123000
vinod        Delhi      20035201470 SB         49000
Anil         Texas      20035201471 SAL        32000
Ching Fu     Chicago    20035201476 SAL        23000
Kiran        Chicago    20035201473 SB         23000
Rocky        Texas      20035201475 SB         13000
```

```
Madhu       Delhi     20035201474 SAL       11000

7 rows selected.

SQL> select cd.cname,at.act_name,cad.actno,cad.act_bal
  2  from cust_dtls cd,act_types_info at,cust_act_dtls cad
  3  where cd.cno=cad.cust_code
  4  and
  5  at.act_type=cad.act_type;

CNAME      ACT_NAME            ACTNO       ACT_BAL
------------ -------------------- ----------- ----------
Anil       Salary A/c.         20035201471      32000
Kiran      Savings Bank A/c.   20035201473      23000
vinod      Savings Bank A/c.   20035201470      49000
vinod      Trading A/c.        20035201472     123000
Madhu      Salary A/c.         20035201474      11000
Rocky      Savings Bank A/c.   20035201475      13000
Ching Fu   Salary A/c.         20035201476      23000

7 rows selected.

SQL> select cd.cname,at.act_name,cad.actno,cad.act_bal
  2  from cust_dtls cd Inner Join cust_act_dtls cad
  3  on cd.cno=cad.cust_code
  4  Inner join  act_types_info at
  5  on cad.act_type=at.act_type;

CNAME      ACT_NAME            ACTNO       ACT_BAL
------------ -------------------- ----------- ----------
Anil       Salary A/c.         20035201471      32000
Kiran      Savings Bank A/c.   20035201473      23000
vinod      Savings Bank A/c.   20035201470      49000
vinod      Trading A/c.        20035201472     123000
Madhu      Salary A/c.         20035201474      11000
Rocky      Savings Bank A/c.   20035201475      13000
Ching Fu   Salary A/c.         20035201476      23000

7 rows selected.
```

Assignment:

1) Write a query to display employee names,salaries ,job titles, hiredate and respective dept names based on salary order?

2) Write a query to display all "salesman and clerk" names,salaries ,job titles, hiredate and respective dept names based on salary and job title order?

3) Write a query to display product detils and company details of each product with min cost 5000 and max cost 40000 also manufactured in current year with warrenty?

4) Write a query to display the customer names, product names , cost , quantity and sales amount generated by the customers from the city "Delhi" and with min sales amount 10000?

5) Write a query to display customer code, customer name, phone_number and actno, act_bal who is maintaining below 5000 act_balance?

6) Write a query to display customer name, Phone number, actno,act_bal and act_name who has opted for loan account?

SELF JOIN

3) SELF JOIN:
A table which is joined itself is known as self join.
In this case we can use alias names for single table.
Here the alias names are temporary.

employee

ename	city
kiran	mumbai
hari	hyd
madhu	hyd
smith	delhi
scott	mumbai
allen	hyd
soumya	chennai
john	delhi

Ex: select * from employee where ename='john'; [NOT CORRECT]

Ex: display emplyoee details who is living in a city where "john" is living?

e1

ename	city
kiran	mumbai
hari	hyd
madhu	hyd
smith	delhi
scott	mumbai
allen	hyd
soumya	chennai
john	delhi-----(matched)

e2

ename	city
kiran	mumbai
hari	hyd
madhu	hyd
smith	delhi(matched)
scott	mumbai
allen	hyd
soumya	chennai
john	delhi(matched)

--goto first alias table and check the employee name "john"
--If it is there then get his "city"

--then the city from first alias table is compared with all city names in 2nd alias table
--If the city values are equal then get the records from second alias table.

```
select e2.ename,e2.city
from employee e1, employee e2
where e1.ename='john'
        and
        e1.city=e2.city;

        (or)

select e2.*
from employee e1, employee e2
where e1.ename='john'
        and
        e1.city=e2.city;
```

output:

```
smith   delhi
john    delhi
```

Ex:

```
select e1.*
from employee e1, employee e2
where e1.ename='john'
        and
        e1.city=e2.city;
```

output: If you display output from First alias table then you will get Duplicate data

```
john    delhi
john    delhi
```

Ex: display the employee details who is working like 'smith'?

```
select e2.* from emp e1, emp e2
where  e1.ename='SMITH' AND e1.job=e2.job;
```

Ex: display customer details who is living in a city where c5 is living?

```
select c2.* from customers c1, customers c2
where c1.cid='c5'  and c1.city=c2.city;
```

```
Ex:--DISPLAY EMPLOYEE DETAILS WHO IS WORKING LIKE "CLARK"
SELECT e2.* from emp e1, emp e2
where e1.ename='CLARK'
  AND
  E1.JOB=E2.JOB;
```

output:

```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
```

OUTER JOINS:

These are used to display ***all data*** from one table and ***only matched data*** from other table.

Types of outer joins: 3

1) Left outer join / left join

Display all the data from left table and only matched data from right table.

2) Right outer join / Right join

Display complete data from right table and only matched data from left table.

3) Full outer join / Full join:

Display
--matched data from both the tables
--unmatched data from left table
--unmatched data from right table

syn-1:

```
select col1, col2, col.....  / *
from table_1  [left join / right join / full join] table_2
ON table1.pk=table2.fk;
```

Ex: display all employee details and if he is working under a dept then display his department details also?

```
select e.*,d.*
from emp e left outer join dept d
ON      e.job='MANAGER' and  e.deptno=d.deptno;
```

Ex: display department details who is having employees with in it?

```
select e.*, d.* from emp e right outer join dept d on e.deptno=d.deptno;
```

Ex:	display emps and their dept details, only emp details who is not working under any dept and only dept details which is not having at least one employee?

```
	select e.*,d.* from emp e full join dept d
	on e.deptno=d.deptno;
```

SAMPLE EXECUTIONS:

```
SELECT * FROM COMP_DTLS;
SELECT * FROM PROD_DTLS;

UPDATE PROD_DTLS SET PROD_COMP_ID='cmp02' WHERE pid='p002';

--Display products information along with company information
/* EQUI JOIN */
SELECT P.*,C.* FROM PROD_DTLS P, COMP_DTLS C
WHERE P.PROD_COMP_ID=C.COMP_ID;

/*LEFT OUTER JOIN */
SELECT P.*,C.* FROM PROD_DTLS P LEFT OUTER JOIN COMP_DTLS C
ON P.PROD_COMP_ID=C.COMP_ID;

/*RIGHT OUTER JOIN */
SELECT P.*,C.* FROM PROD_DTLS P RIGHT OUTER JOIN COMP_DTLS C
ON P.PROD_COMP_ID=C.COMP_ID;

/*FULL OUTER JOIN */
SELECT P.*,C.* FROM PROD_DTLS P FULL OUTER JOIN COMP_DTLS C
ON P.PROD_COMP_ID=C.COMP_ID;
```

SAMPLE OUTPUTS:

```
COMP_ID COMP_NAME            COMP_COUNTRY
------- -------------------- --------------------
cmp01   sony                 japan
cmp02   wipro                india
cmp04   semantic             usa

PID  PNAME                      COST MFG       WARRENTY   PROD_COMP_ID
---- -------------------- ---------- --------- ---------- ------------
p001 smart phone               34000 12-MAY-14 1 year     cmp01
p002 laptop                    54000 03-FEB-14 3 years    cmp01
p005 Mobile                    24000 08-AUG-14 1 year
p007 Desktop                   24000 22-OCT-10 2 year

1 rows updated.
PID  PNAME                      COST MFG       WARRENTY   PROD_COMP_ID
---- -------------------- ---------- --------- ---------- ------------
p001 smart phone               34000 12-MAY-14 1 year     cmp01
```

```
p002 laptop                54000 03-FEB-14 3 years    cmp02
p005 Mobile                 24000 08-AUG-14 1 year
p007 Desktop                  24000 22-OCT-10 2 year

PID  PNAME                    COST MFG      WARRENTY   PROD_COMP_ID COMP_ID
COMP_NAME           COMP_COUNTRY
---- -------------------- ---------- --------- ---------- ----------- ------- ------------------- --------------------
p001 smart phone              34000 12-MAY-14 1 year     cmp01       cmp01   sony
japan
p002 laptop                54000 03-FEB-14 3 years    cmp02       cmp02   wipro
india

PID  PNAME                    COST MFG      WARRENTY   PROD_COMP_ID COMP_ID
COMP_NAME           COMP_COUNTRY
---- -------------------- ---------- --------- ---------- ----------- ------- ------------------- --------------------
p001 smart phone              34000 12-MAY-14 1 year     cmp01       cmp01   sony
japan
p002 laptop                54000 03-FEB-14 3 years    cmp02       cmp02   wipro
india
p005 Mobile                 24000 08-AUG-14 1 year
p007 Desktop                  24000 22-OCT-10 2 year

PID  PNAME                    COST MFG      WARRENTY   PROD_COMP_ID COMP_ID
COMP_NAME           COMP_COUNTRY
---- -------------------- ---------- --------- ---------- ----------- ------- ------------------- --------------------
p001 smart phone              34000 12-MAY-14 1 year     cmp01       cmp01   sony
japan
p002 laptop                54000 03-FEB-14 3 years    cmp02       cmp02   wipro
india
                                              cmp04   semantic            usa

PID  PNAME                    COST MFG      WARRENTY   PROD_COMP_ID COMP_ID
COMP_NAME           COMP_COUNTRY
---- -------------------- ---------- --------- ---------- ----------- ------- ------------------- --------------------
p001 smart phone              34000 12-MAY-14 1 year     cmp01       cmp01   sony
japan
p002 laptop                54000 03-FEB-14 3 years    cmp02       cmp02   wipro
india
p005 Mobile                 24000 08-AUG-14 1 year
p007 Desktop                  24000 22-OCT-10 2 year
                                              cmp04   semantic            usa
```

Assignments:

TABLES:

```
cust_dtls ( cid, cname, city, gender,mobile)
            PK
act_types ( act_type, act_name, act_desc)
                 PK

Cust_act_dtls ( actno, act_type, act_open_date, act_bal, custid)
                 PK       FK                                   FK
```

Ex: Display customer name, actno, act type and act name?

Ex: Display Female customers from CHICAGO, with their accounts information who have DEMAT account?

Ex: Display all customer details and if a customer having SALARY account then display account details also?

Ex: Display all customers personnel details and all accounts list of information?
If a customer have an account then display his details along with his account details?

--Display customer account details who is maintaining an account similar to " kiran "

```
select cad2.*
from cust_act_dtls cad1, cust_act_dtls cad2,cust_dtls cd
where cd.cname='Kiran' and
    cd.cno=cad1.cust_code
    and
    cad1.act_type=cad2.act_type;
```

SUB QUERIES

A query with in other query is known as sub query.
Sub queries are preferable to display output from one table and having an input value from other table.

Syntax:

select ..... from table...where [=/IN/exists/not exists](select ..... from table......where ....[=/IN/exits/not exists](select.......).....);

Execution process:
Here the execution process is always from innermost query to the outermost.

Outer query<-----------o/p<------------Inner query<----------o/p<---------Inner query

TYPES OF SUBQUERIES: 2

1) Single row sub query:

A sub query which returns single output value.
In this case in between the outer and inner queries we can use = operator.

2) Multi row sub query:

A sub query which returns multiple output values.
In this case in between the outer and inner queries we can use IN operator.

*****CORRELATED SUBQUERY:**

A sub query which depends on a value generated by outer query. Here the execution process is as follows.

```
outer query-------o/p-------> Inner query-------->o/p---------->
                                                                |
        ^                                                       |
        |                                                       V
        <----------------------------------------------------
```

- First Outer query has to be executed and generates some output values
- Second based on these output values, sub query has to be executed
- From sub query we have some output and it is passed to outer query again
- Then outer query has to be executed again.

Ex: display department details of employee smith?

```
select * from dept
where deptno IN(select deptno from emp where ename='SMITH');
```

By using join Query:

```
select d.* from emp e, dept d
where e.ename='SMITH' and e.deptno=d.deptno;
```

Ex: Display employee details who is working under PRODUCTION?

```
        select * from emp where deptno=(select deptno from dept where
dname='PRODUCTION');
```

Ex: display the department details of all managers?

```
        select * from dept
        where deptno IN(select deptno from emp where job='MANAGER');
```

Ex: display accounting department employee details?

```
        select * from emp where deptno=(select deptno from dept where
dname='ACCOUNTING');
```

sample tables:

```
                    comp_dtls
                    ---------
        comp_code   comp_name   comp_country
        (PK)
         |
         |
         V---------------------------------------------------------------------------------->
                                                                                           |
                                                                                           |
                                                                                           |
                    prod_dtls                                                              |
                    ---------                                                              V
        prodid  prodname    prodcost    mfg         warrenty    prod_comp_code
        (PK)                                                    (FK)
```

Ex: display product details from the company sony?

```
        select * from prod_dtls where prod_comp_code=(select comp_code from comp_dtls
where comp_name='sony');
```

Ex: display product details from the companies other than sony, samsung?

```
        select * from prod_dtls where prod_comp_code IN(select comp_code from comp_dtls
where comp_name NOT IN('sony','samsung'));
```

Ex: display company names of the prod ids p003,p007?

```
select comp_name from comp_dtls where comp_code IN( select prod_comp_code
from prod_dtls where prodid IN('p003','p007'));
```

CORRELATED SUBQUERIES:

These sub queries use 2 operators either EXISTS OR NOT EXISTS

EXISTS
It returns true if a sub query fetches at least one value. If it returns TRUE then outer query will display the result.

NOT EXISTS
It returns true if a sub query fetches no values at all. If it returns TRUE then outer query will display the result.

Ex: display department details which is having at least one employee with in it?

```
select d.* from dept d
where exists( select 10  from emp e where e.deptno=d.deptno);
```

Ex: display department details which not having at least one employee with in it?

```
select d.* from dept d
where not exists( select 10  from emp e where e.deptno=d.deptno);
```

ex: Display compnies information from we are maintaining at least one product from each company?

Ex: Display company details from which we have no products at all?

Examples:

--Display department details of empid 7788?

```
select d.* from emp e, dept d
where e.empno=7788 and e.deptno=d.deptno;
```

```
select * from dept where deptno=(select deptno from emp where empno=7788);
```

--Display employee details working under ACCOUNTING dept?

```
select * from emp where deptno=(select deptno from dept where dname='ACCOUNTING');
```

--Display empno,ename,sal from the departments ACCOUNTING, RESEARCH based on emp sal order?

```
select empno,ename,sal from emp where deptno in(select deptno from dept where dname
IN ('ACCOUNTING','RESEARCH'))
ORDER BY SAL;

--Display customer account details who is from the cities Delhi and Chicago?

select * from cust_act_dtls where cust_code IN(select cno from cust_dtls where city
in('Delhi','Chicago'));

--Display customer details who is maintaining Savings Bank A/c. ?

select * from cust_dtls
where cno IN(select cust_code from cust_act_dtls
                where act_type =(select act_type from act_types
                                where act_name='Savings Bank A/c.'));
```

VIEWS

Views are known as logical tables. They represent the data of one of more tables. A view derives its data from the tables on which it is based. These tables are called base tables. Views can be based on actual tables or another view also.

QUERY EXECUTION PROCESS

```
                         <---------------------DATA BASE SERVER----------------------->

CLIENT/USER              ORACLE/DB ENGINE                      DATABASE


Request                  DATABASE
( query  )------------->ENGINE--------------->--^--------------->DB
                           |             |             |
                         sql stmt executor |             |
                           |             |            Data dictionary( meta data )
                         pl/sql engine   |             |
                           |             |             |
                           |--------------------      Actual Data
                                                       |
                                                     ---------------------
                                                     |           |
                                   selecting from    |           |
                                   table             V           |
                                   1<------------------------    |
                                   2<------------------------    |
                                   3<------------------------    |
                                   4<------------------------    |
                                        : n hits to DB           |
                                   n<------------------------    |
                                                                 |
                                                                 |
                                                                 |
                         output    selecting from view           |
                         <---------CACHE    <---------------------------------------
                         <---------MEMORY   Only one HIT to the DB
                         <---------
                         <---------
                           :
                         <---------
```

View:

It is a database object contains logical copy of selected table data.
It can be created based on frequently using data.
It reduces number of hits to the database.
It improves the performance of queries and database.

Types of Views: 2

1) Simple view / updateable view:
It is created based on single table data. It allows dml operations.

syn:
```
create view  <view_name>
AS  select ...... from table_name where <cond>;
```

2) Composite view or read only view:
It is created based on morethan one table data. It doesnt allow dml operations.

Advantage: It will reduce writing number of join queries again and again.

Syntax:
```
create view <vw_name>
AS select ........ from  table1, table2,..... where <join-cond>;
```

NOTE:
By default the **Client user** not having permission to create views. That can be assigned by DBA.

Ex: write a query to create a view which contains managers information from emp table?

```
create view  vw_mgr_info
as select * from emp where job='MANAGER';
```

Error: Insufficient privillages

ex: connect to dba

```
system
manager
orcl
```

Ex:
```
grant create view to manju;

grant succeeded
```

Ex:
```
conn manju /welcome@orcl
connected.
```

Ex:
```
create view  vw_mgr_info
as select * from emp where job='MANAGER';
```

view created.

HOW TO SELECT THE DATA FROM VIEW?

SYN: SELECT * from <vw_name>;

Ex: select * from vw_mgr_info;

sample output:

```
 EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
```

Ex: display manager names and salaries?

select ename,sal from vw_mgr_info;

```
ENAME             SAL
---------- ----------
JONES            2975
BLAKE            2850
CLARK            2450
```

--SUB TABLES MAINTAINS DATA STATICALLY
--VIEWS MAINTAINS DATA DYNAMICALLY.

sample hands-on:

sub table:

create table emp_mgr as select * from emp where job='MANAGER';

Ex:
select * from emp_mgr;

```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      1122 dinesh     MANAGER              12-MAY-13       4500        700         40
```

Ex: creating a view containing managers info?

```
create view vw_mgr_info as select * from emp where job='MANAGER';
```

Ex:
```
select * from vw_mgr_info;

    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER        7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER        7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER        7839 09-JUN-81       2450                    10
      1122 dinesh     MANAGER             12-MAY-13       4500        700         40
```

Ex:
```
insert into emp values(2233,'naresh','MANAGER',NULL,'21-MAY-14',3400,200,10);

1 rows inserted.
```

Ex:

```
select * from emp_mgr;

    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER        7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER        7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER        7839 09-JUN-81       2450                    10
      1122 dinesh     MANAGER             12-MAY-13       4500        700         40


select * from vw_mgr_info;

    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER        7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER        7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER        7839 09-JUN-81       2450                    10
      1122 dinesh     MANAGER             12-MAY-13       4500        700         40
      2233 naresh     MANAGER             21-MAY-14       3400        200         10
```

Ex:
```
delete from vw_mgr_info where empno in(1122,2233);

select * from emp_mgr;
select * from vw_mgr_info;
select * from emp where job='MANAGER';


    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER        7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER        7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER        7839 09-JUN-81       2450                    10
      1122 dinesh     MANAGER             12-MAY-13       4500        700         40
```

```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
```

2) READ ONLY VIEWS / COMPOSITE VIEWS:

Ex: write a query to maintain a view containing managers ename, sal, job, dept name and dept loc?

```
        create view vw_emp_dept_info
        as select ename,sal,job,dname,loc from emp,dept
                                where job='MANAGER'  and  emp.deptno=dept.deptno;
```

Ex: Display manager details and their dept details?

```
select * from VW_EMP_DEPT_INFo;

ENAME             SAL JOB       DNAME          LOC
---------- ---------- --------- -------------- -------------
JONES            2975 MANAGER   RESEARCH       DALLAS
BLAKE            2850 MANAGER   SALES          CHICAGO
CLARK            2450 MANAGER   ACCOUNTING     NEW YORK
```

Ex: display manager names and their department names?

```
        select ename,dname from vw_emp_dept_info;


ENAME      DNAME
---------- --------------
JONES      RESEARCH
BLAKE      SALES
CLARK      ACCOUNTING
```

3) Inline view

An inline view is a SELECT statement in the FROM-clause of another SELECT statement. In-line views are commonly used to simplify complex queries by removing join operations and condensing several separate queries into a single query.

Ex: sub query

Generally sub query is to be written inside the WHERE clause.

Display employee details working under accounting department?

```
        select * from emp where deptno=( select deptno from dept where
dname='ACCOUNTING');
```

Ex: Inline view

```
SELECT *
  FROM ( SELECT deptno, count(*) emp_count
         FROM emp
         GROUP BY deptno ) emp,
       dept
 WHERE dept.deptno = emp.deptno;
```

Explanation:

```
        deptno          emp_count
        ------          ---------
        10              4
        20              6
        30              7
```

```
Ex:     deptno          dname           loc        emp_count
        ------          -----     -     ---        ----------
        10              Accounting      chicago                 4

        20              REsearch        Texas           6

        30              Sales           Dallas          7
```

Ex: display the employees who earn the highest salary in each department?

```
SELECT *
  FROM
       ( SELECT deptno, max(sal) maxsal
         FROM emp
         GROUP BY deptno ) b, emp a
 WHERE
a.sal= b.maxsal
and
a.deptno = b.deptno;
```

output:

```
DEPTNO     MAXSAL      EMPNO ENAME      JOB             MGR HIREDATE        SAL
COMM     DEPTNO
---------- ---------- ---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      30       2850       7698 BLAKE      MANAGER         7839 01-MAY-81       2850
30
      20       3000       7788 SCOTT      ANALYST         7566 19-APR-87       3000
20
      10       5000       7839 KING       PRESIDENT            17-NOV-81       5000
10
      20       3000       7902 FORD       ANALYST         7566 03-DEC-81       3000
20
```

Ex:

```
SELECT d.dept_id, d.name, emp_cnt.tot

FROM department d INNER JOIN

 (SELECT dept_id, COUNT(*) tot

  FROM employee

  GROUP BY dept_id) emp_cnt

  ON d.dept_id = emp_cnt.dept_id;
```

```
   DEPT_ID NAME                        TOT

---------- -------------------- ----------

        10 ACCOUNTING                    3

        20 RESEARCH                    5

        30 SALES                     6
```

Ex: Display comp_codes, number of products from each company code and also display company details?

```
select  * from (select comp_code,count(*) from prod_dtls group by comp_code) p,comp_dtls
C
where p.comp_code=c.comp_code;
```

ex:

```
select d.deptno,d.dname,d.loc,e.deptno,count(e.empno) from emp e ,dept d
where e.deptno=d.deptno
group by d.deptno,d.dname,d.loc,e.deptno;
```

Creating FORCE VIEWS:

A view can be created even if the defining query of the view cannot be executed, as long as the CREATE VIEW command has no syntax errors. We call such a view a view with errors. For example,

if a view refers

--to a non-existent table or

--an invalid column of an existing table, or

--if the owner of the view does not have the required privileges,

then the view can still be created and entered into the data dictionary.

You can only create a view with errors by using the FORCE option of the CREATE VIEW command:

CREATE FORCE VIEW AS ...;

Ex:

```
create FORCE view vw_act_dtls_Sb
as
select * from cust_act_dtls where act_type='SB' order by act_bal;
```

Materialized Views in Oracle

A materialized view, or snapshot as they were previously known, is a table segment whose contents are periodically refreshed based on a query, either against a local or remote table. Using materialized views against remote tables is the simplest way to achieve replication of data between sites.

```
CREATE MATERIALIZED VIEW emp_mv

BUILD IMMEDIATE

REFRESH FORCE

ON DEMAND

AS

SELECT * FROM emp@db1.world;
```

INDEXES

Index is a table like object which maintains ordered data of the column physically.
It reduces number of comparisons to make, to fetch the required data. It fastens the search as much as fast. It occupies physical disk space.

Index contains 2 parts
i) Data part ii) Address part (**ROWID**)

ROWID is a pseudo column / virtual column. It contains Physical address of each record. We can access the values of rowid , but we cannot modify them.

This (**ROWID**) address is a combination of the following.

{fileno datablock_no record_no}.

This first data block of a table is indicated with 0.
And the first record in each data block is indicated with 0.

Ex: let us consider the file [table] number is 555.

select * from prod_dtls;

```
555.000.000
555.000.001
555.000.002
      :
555.000.099-----> this is the last record in first data block of table no 555. (100 records)

555.001.000
555.001.001
555.001.002
  :     :
555.001.999------> this is the last record in second data block of table no 555 ( 1000 records)

555.002.000
555.002.001
     :
555.002.009------> this is the last record in the third data block of table no 555
```

ex:

salaries(table)	Idx_sal(indexed table)
1--9000	2--1000
2--1000	5--1000
3--2000	8--1000
4--9000	3--2000
5--1000	7--2000
6--2500	9--2000
7--2000	6--2500
8--1000	1--9000
9--2000	4--9000
10--9000	10-9000

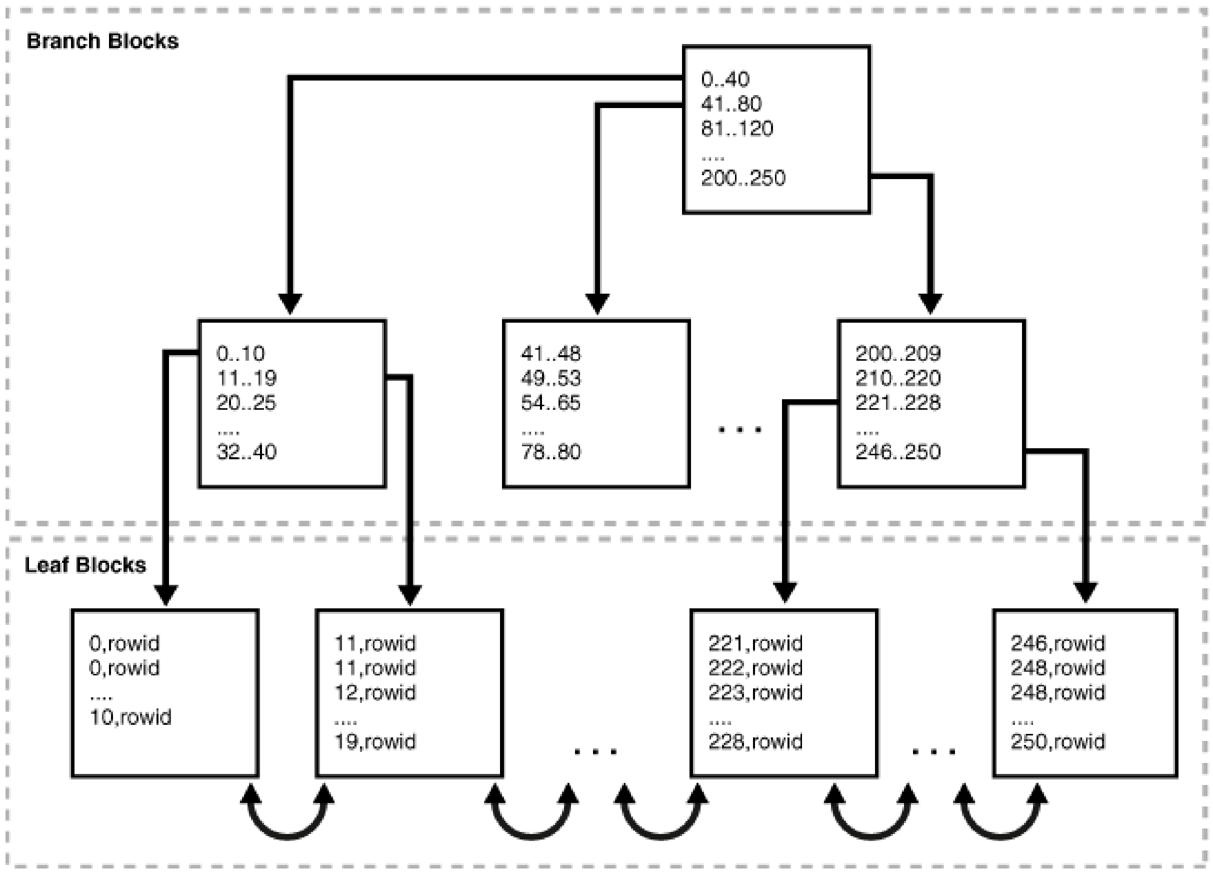
Before INdex creation:	After INdex creation:
select * from emp where sal <=2000;	select * from emp where sal <=2000;
10 comparisions	7 comparisions
6 salaries	6 records

B-Tree Indexes

B-trees, short for **balanced trees**, are the most common type of database index. A B-tree index is an ordered list of values divided into ranges. By associating a key with a row or range of rows, B-trees provide excellent retrieval performance for a wide range of queries, including exact match and range searches.

Figure 3-1 illustrates the structure of a B-tree index. The example shows an index on the department_id column, which is a foreign key column in the employees table.

Figure 3-1 Internal Structure of a B-tree Index



TYPES OF INDEXES: 2

1) Simple index:
It is created on a table on single column.

Syntax:
```
create index <idx_name>
on table_name(colname);
```

2) Composite index
It is created on multiple columns of a table.

Syntax:
```
create index <idx_name>
on table_name(col1 ,col2,....);
```

Ex: create an index on table emp on the column sal?

```
create index idx_sal on emp(sal);
```

Ex: create an index on prod_dtls on the columns cost,comp_code?

```
create index idx_prod_cost_comp_code on prod_dtls(cost,comp_code);
```

3) Function Based Index
In the index definition, If we specify any **calculations or functions on table columns** then it is known as Function Based Index.

syn:
```
create index  <idx_name>
on table_name( col+100,col2*0.10,Func(col3));
```

Ex:
```
create index idx_sal_ename
on emp((0.20*sal),Initcap(ename));
```

Ex:
```
select sal,(0.20*sal),initcap(ename) from emp;
```

REVERSE KEY INDEX

In this index , the search criteria is from right most leaf to left.
This index is preferable to search for highest values frequently.

Syntax: create index <idx_name> on table_name(col) ***REVERSE***;

Ex: create a reverse index on sales transaction table on sales amount column?

```
Create index idx_higher_sales on sales(sales_amt) REVERSE;
```

UNIQUE INDEX
If an index is creating on Unique column then it is known as unique index.

Ex:

```
create unique index idx_comm
on emp(comm);
```

Bitmap Index
Use this index if you have very less number of different values in a column like gender, emp_job_status

Ex:

```
create bitmap index ix_gender
on cust_dtls(gender);
```

Note
By default the oracle engine create and maintains a default index on each primary key column of a table.

Note

In Oracle indexes information is maintained under a system defined table called ***USER_INDEXES***

HOW TO DELETE AN INDEX?

drop index idx_name;

CLUSTERS

Creating Clusters

To create a cluster in your schema, you must have the **CREATE CLUSTER** system privilege and a quota for the tablespace intended to contain the cluster or the **UNLIMITED TABLESPACE** system privilege.

To create a cluster in another user's schema you must have the **CREATE ANY CLUSTER** system privilege, and the owner must have a quota for the tablespace intended to contain the cluster or the **UNLIMITED TABLESPACE** system privilege.

You create a cluster using the **CREATE CLUSTER** statement. The following statement creates a cluster named **emp_dept**, which stores the **emp** and **dept** tables, clustered by the**deptno** column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
```

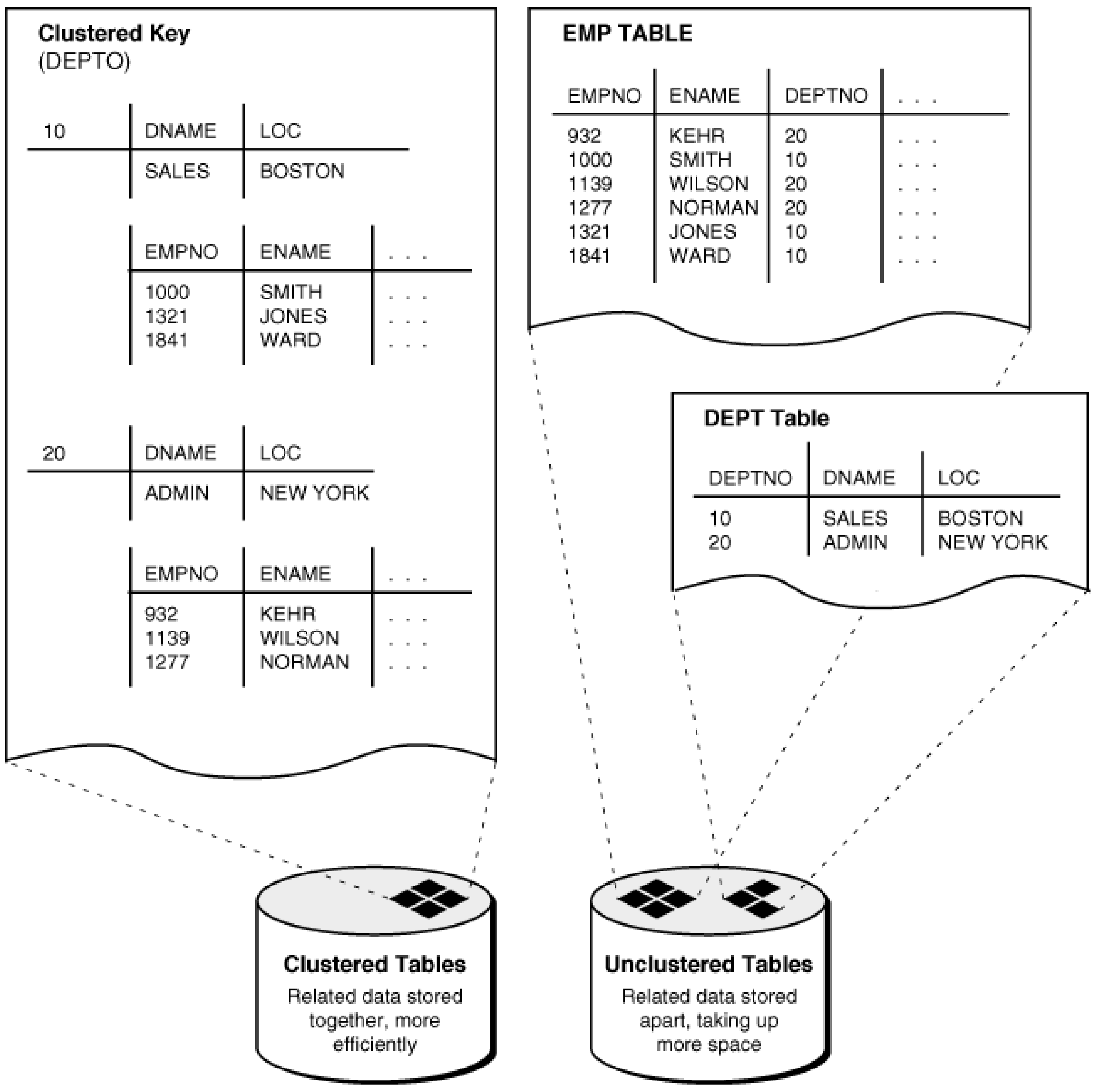
If no **INDEX** keyword is specified, as is true in this example, an index cluster is created by default. You can also create a **HASH** cluster, when hash parameters (**HASHKEYS**, **HASH IS**, or**SINGLE TABLE HASHKEYS**) are specified

Creating Clustered Tables

To create a table in a cluster, you must have either the **CREATE TABLE** or **CREATE ANY TABLE** system privilege. You do not need a tablespace quota or the **UNLIMITED TABLESPACE**system privilege to create a table in a cluster.

You create a table in a cluster using the **CREATE TABLE** statement with the **CLUSTER** clause. The **emp** and **dept** tables can be created in the **emp_dept** cluster using the following statements:

```
CREATE TABLE emp (

  empno NUMBER(5) PRIMARY KEY,

  ename VARCHAR2(15) NOT NULL,

  . . .

  deptno NUMBER(3) REFERENCES dept)

  CLUSTER emp_dept (deptno);


CREATE TABLE dept (

  deptno NUMBER(3) PRIMARY KEY, . . . )

  CLUSTER emp_dept (deptno);
```



Cust_dtls (custid,cname,city)

Act_types (act_type,act_name,desc)

Cust_act_dtls(actno,act_type,act_open_dt,act_bal,custid)

Example:

```
create cluster emp_dept (deptno  number(2));


create table deptcp

(

deptno  number(2) primary key,

dname  varchar2(20),

loc  varchar2(20)

)

cluster emp_dept(deptno);


create table empcp

(

eid  number(4) primary key,

ename varchar2(20),

sal  number(5),

deptno number(2) references deptcp(deptno)

)

cluster emp_dept(deptno);
```

SEQUENCES

Ex:	1 2 3 4 5 ....

	10 20 30 40......

	1000 1001 1002.....

	20035201000 20035201001 20035201002

Ex:
```
create sequence actno_serial
start with 20035201000;

sequence created.
```

SEQUENCES

It is a data base object which is used to generate sequential integers with the specified interval value.
Generally the sequences are used to generate primary key values.

syn:

```
create sequence <seq_name>
increment by <val>
start with <val>
max value <val>;
```

Note: by default the sequence starts with 1 and increment value is also 1.

** Sequence is an independent object.(It is not depending on any table)

Ex: create sequence srno1;

Pseudo columns:

<seq_name>.CURRVAL	Display current value of the sequence
<seq_name>.NEXTVAL	Display nextvalue of sequence

Ex: select srno1.currval from dual;

```
CURRVAL
----------
       1
```

Ex: select srno1.nextval from dual;

```
NEXTVAL
----------

2
```

Ex: create a sequence starts with 1001?

create sequence custno start with 1001;

Ex: update the cust_dtls table under that update the cust_srno column with sequence values?

update cust_dtls set cust_srno=srno1.nextval;

Ex: adding a primary key constraint on the column cust_srno?

```
alter table cust_dtls add constraint pk_custsrno primary key(cust_srno);
```

Ex: Insert new record in the table cust_dtls along with sequence values?

```
insert into cust_dtls values(4,'abc','hyd',6767676765,'F',null,srno1.nextval);
                                                             **************
```

Hands-on:

```
CREATE TABLE CUSTOMERS
(
ID INT,
NAME  VARCHAR2(20),
SRNO  NUMBER)

CREATE SEQUENCE ID START WITH 6600 INCREMENT BY 5
CREATE SEQUENCE SRNO

INSERT INTO CUSTOMERS VALUES(ID.NEXTVAL,'AJAY',NULL)
INSERT INTO CUSTOMERS VALUES(ID.NEXTVAL,'KIRAN',NULL)
INSERT INTO CUSTOMERS VALUES(ID.NEXTVAL,'MADHU',NULL)

SELECT * FROM CUSTOMERS

UPDATE CUSTOMERS SET SRNO=SRNO.NEXTVAL
```

results:

```
CREATE TABLE succeeded.
CREATE SEQUENCE succeeded.
CREATE SEQUENCE succeeded.
1 rows inserted
1 rows inserted
1 rows inserted
ID                   NAME                 SRNO
-------------------- -------------------- --------------------
6605                  AJAY
6610                  KIRAN
6615                  MADHU

3 rows selected

3 rows updated
ID                   NAME                 SRNO
-------------------- -------------------- --------------------
6605                  AJAY                 1
6610                  KIRAN                 2
6615                  MADHU                   3

3 rows selected
```

Ex:

```
CREATE SEQUENCE t1_seq;

CREATE TABLE t1 (
  id          NUMBER DEFAULT t1_seq.NEXTVAL,
  description VARCHAR2(30)
);

INSERT INTO t1 (description) VALUES ('DESCRIPTION only');
INSERT INTO t1 (id, description) VALUES (999, 'ID=999 and DESCRIPTION');
INSERT INTO t1 (id, description) VALUES (NULL, 'ID=NULL and DESCRIPTION');

SELECT * FROM t1;

        ID DESCRIPTION
---------- ------------------------------
         1 DESCRIPTION only
       999 ID=999 and DESCRIPTION
           ID=NULL and DESCRIPTION

3 rows selected.
```

HOW TO DELETE THE SEQUENCE?

Ex: `drop sequence srno;`

system table:

user_sequences

SYNONYMS

Synonyms are used to create permanent alias names for the tables.

Types of Synonyms: 2

I) Private synonym
It is a default synonym. and it is used by the owner only.

ii) Public synonym
It is created by DBA. And it can be accessed by permitted authenticated user.

syn: `create [ public ] synonym <syn_name> FOR <table_name>;`

Ex: `create synonym cd for cust_dtls;`

Note:

Once a synonym is created then we can use synonym name instead of table name for any operations on the table.

Ex: display cust details data?

```
select * from cust_dtls;

        or

select * from cd;
```

Ex:

```
insert into cd values((10,'c','mumbai',null,'M',null,custno.nextval);
```

Ex: deleting the data from table using alias name?

```
delete from cd where mobile is null;
```

Ex: how to delete a synonym?

```
drop synonym cd;
```

ORACLE'S SQL FUNCTIONS

These functions are predefined and used to perform user required operations.

CATEGORES: 2

1) GROUP OR AGGREGATE FUNCTIONS
These functions can acts on group of values display single output value. These functions act on column level / field level.

Ex: sum(), avg()...

2) SCALAR /SINGLE ROW FUNCTIONS
These functions acts on group of values and display a set of output values. These functions act on Record level / row level.

Ex: lower(), length(), trim()

The above 2 categories of functions are further divided into following 4 types.

TYPES OF FUNCTIONS: 4

i) Numeric functions
Acts on numeric data

ii) String functions
Acts on char data

iii) Date functions

Acts on date data

iv) Conversion functions

It acts on one data type and display result in other data type.

NUMERIC FUNCTIONS [GROUP FUNCTIONS]

These functions act on column data only.

i) SUM(colname)

It will display addition of values from the column

Ex: display addition of all salaries?

```
select sum(sal) from emp;
```

Ex: Find the sum of salaries for managers?

Ex: Find the sum of salaries of emps working under RESEARCH department?

Ex: Find the total investment amount for the products from sony?

Ex: Find the total balance from the account types SB and SAL?

ii) AVG(colname)

It will display average value from the column

Ex: display average product cost?

```
select avg(cost) AvgCostOfProduct
from prod_dtls;
```

Ex: Find the average sales amount in the last year?

```
select avg(sales_amount) from sales
where tid like'%14';
```

iii) MAX(colname)

Display higher value from the column.

Ex: Display highest salary among all salesman?

```
select max(sal ) from emp where job='SALESMAN';
```

Ex: display employee details who is getting higher salary?

```
select * from emp where sal=max(sal);-----WRONG
          Or

select * from emp where sal IN(select max(sal) from emp);
```

iv) MIN(colname)

Display lower value from the column

Ex: display least cost product details among all mobiles?

```
select * from prod_dtls where cost IN(select min(cost) from prod_dtls where pname='MOBILE');
```

v) COUNT(colname)

Display number of values from the column except null values

Ex: find how many emps getting commission?

```
select count(empno) from emp where comm is not null;

select count(comm) from emp;
```

vi) COUNT(*)

Display number of records from a table.

Ex: display number of transactions on current day?

```
select count(*) from trans_dtls where trans_date=sysdate;
```

Note:

" Aggregate functions executes on field /column level data "

Ex: Display number of emps under deptno 10?

```
select count(*) from emp where deptno=10;

10      5
```

Ex: Display number of emps under deptno 20?

```
select count(*) from emp where deptno=20;

20      7
```

Expected output:

```
10      5
20      7
30      5
40      2
50      1
```

GROUP BY Clause:

It is used to group related data by considering distinct values from the column. On each group the oracle engine executes the aggregate function and display result individually.

Syntax:

```
select  colname, colname,..., aggfunction1, aggregate2,.....
from table
where <cond>
GROUP BY <col1>,<col2>,...
HAVING aggfunc1,...
ORDER BY  cl1,col2,.....;
```

Ex: find out number of emps working under each dept ?

```
select  deptno, count(*) " No. of emps"
from emp
GROUP BY deptno;
```

output:

```
deptno              No. of emps
---------       -------------
30                  6
20                  7
10                  5
```

Ex: find out number of emps working under each dept on order of deptno?

```
select  deptno, count(*) " No. of emps"
from emp
GROUP BY deptno
order by deptno;
```

output:

```
10      5
20      7
30      6
```

Ex: findout max sal under each deptno?

Ex: findout min sal for each job category based on the order of job?

Ex: findout number of emps under each dept getting morethan 5000 salary based on the order of deptno?

```
select  deptno,  count(*)
from emp
```

```
where sal> 5000
group by deptno
order by  deptno;
```

Ex: find out number of customers from each city based on order of city?

Ex: find out number of customers for each account type?

Ex find out number of products from each product category?

HAVING clause:

It is used to specify conditions on group by output.

Ex: find out number of emps working under each dept on order of deptno if a dept contains at least 10 emps?

```
select  deptno, count(*) " No. of emps"
from emp
GROUP BY deptno
HAVING count(*)>=10
order by deptno;
```

```
output: 20      70
        30      60
```

Examples:

```
select deptno,count(empno) EmpCount from emp group by deptno;

select deptno,count(empno) EmpCount
from emp
group by deptno
Having count(empno)>=5 ;

select job,count(empno) EmpCount
from emp
group by job;
--Having count(emEx:  Display number of emps under deptno 10?

        select count(*) from emp where deptno=10;pno)>=5 ;

select job,count(empno) EmpCount
from emp
group by job
Having count(empno)<=3 ;


select d.dname,avg(e.sal) "Average Salary" from dept d, emp e
```

```
where d.deptno=e.deptno
group by d.dname;
```

OUTPUT:

```
 DEPTNO   EMPCOUNT
---------- ----------
       30        6
       20        5
       10        2

   DEPTNO   EMPCOUNT
---------- ----------
       30        6
       20        5
       10        2

   DEPTNO   EMPCOUNT
---------- ----------
       30        6
       20        5

JOB        EMPCOUNT
--------- ----------
CLERK            4
SALESMAN         4
PRESIDENT        1
MANAGER          2
ANALYST          2

JOB        EMPCOUNT
--------- ----------
PRESIDENT        1
MANAGER          2
ANALYST          2

DNAME          Average Salary
-------------- --------------
ACCOUNTING               3150
RESEARCH                 2175
SALES                    1925
```

NUMERIC FUNCTIONS (SCALAR FUNCTIONS)

These functions are acting on record level.

1) **ABS(n) [ABSOLUTE]**
   Display absolute value of n.

Ex: select abs(-9) from dual;

9

Ex: select abs(13.23) from dual;
13.23

Ex: select abs(round((months_between(hiredate,sysdate)/12))) from emp;

2) **mod(m,n)**
Display remainder value after m devides n.

Ex: select mod(25,5) from dual;
0

Ex: select mod(17,3) from dual;
2

3) **power(m,n)**
Display m power nth value

Ex: select power(5,3) from dual;
125

4) **SQRT(n) [square root]**
Display square root value of n

Ex: select sqrt(64) from dual;
8

5) **ROUND(m,n)**
Display value *"m"* which is rounded to the *"n"* number of decimal places.
Before displaying *"n th"* Decimal digit it will check *" n+1 th"* decimal digit, if it is > or = 5 then *"nth"* digit incremented by 1.

Ex: select round(63.354,2) from dual;
63.35

Ex: select round(63.354,1) from dual;
63.4

Ex: select round(63.354) from dual;
63

Ex: select round(69.554) from dual;
70

6) **TRUNC(m,n)**
Display value m which is truncated to the n number of decimal places.

Ex: select trunc(63.354,1) from dual;
63.3

Ex: select trunc(69.554) from dual;

69

7) FLOOR(n)
Display highest integer value which is lessthan or equal to given value.

Ex: select floor(64.2) from dual;
{0,1,2,......,61,62,63,64}= 64

8) CEIL(n)
Display lowest integer value which is greater than or equal to given value.

Ex: select ceil(64.2) from dual;
{65,66,67,,,........}=65

9) LEAST(val/expr,val/expr,....)
Display minimum value from the given values or expression results.

Ex: select least(32,(6*5), (20-10), (36/2)) from dual;
10

10) GREATEST (val/expr, val/expr,.....)
Display maximum value from the given values or expressions

Ex: select greatest (32,(6*5), (20-10), (36/2)) from dual;
32

STRING FUNCTIONS (scalar functions)

1) ASCII('ch')
display ascii value of the character

Ex: select ascii('a') from dual;
97

Ex: select ascii('A') from dual;

65

Ex: select ascii('@') from dual;
64

2) LENGTH('str'/col)
Display number of chars from the given string or column values

Ex: select ename, length(ename) " length of name" from emp;

Ex: select length('oracle') from dual;

6

3) LOWER('str'/col)
Display the given string chars or column values in lower case.

Ex: select lower(ename) from emp;

Ex: select lower('HAI') from dual;

hai

4) UPPER('str'/col)
Display given string chars or column values in upper case

Ex: select upper (pname) from products;

5) INITCAP('str'/col) [initial capital]
Display the given string or column values with begining char as capital.

Ex: select initcap('welcome to oracle') from dual;

Welcome To Oracle

6) SUBSTR('str'/col,m,n) (substring)
Display a substring from the given string. Here the substring started with "m" th char and through "n" number of chars.

Ex: select substr('secure',3,4) from dual;
cure

Ex: select substr('welcome to oracle functions',12,6) from dual;
oracle

Ex: select substr('welcome to oracle functions',12) from dual;
oracle functions

7) INSTR('str'/col,'ch',m,n) [instring]
Display the position of char in the given string or col.

Here "m" value is either +1(default), or -1
+1 Means search the character position from the begining of string.
-1 Means search the character position from the end of string.

Here "n" is nth occurance of give character.

ex: select instr('welcome','e') from dual;
2
or

ex: select instr('welcome','e',1,1) from dual;
2

Ex: display second occurance of 'e' from the begining of string?

select instr('welcome','e',1,2) from dual;

7

Ex: Find the length of username in a mail id?

```
select email,  instr(email,'@') - 1 " Length of Mail"   from emp;
```

Ex: `select instr('dineshp.ora@gmail.com','@')-1  from dual;`

output:- 11

select instr('welcome','e') from dual; O/P: 2

select instr('welcome','e',+1) from dual; O/P: 2

select instr('welcome','e',-1,1) from dual; O/P: 7

select instr('welcome','e',-1,2) from dual; O/P: 2

8) TRANSLATE('str'/col, 'sourcechars','targetchars')

It will display given string chars by translating source chars with corresponding target chars.

Ex:
```
select  translate('welcome','em','xy')  from dual;
wxlcoyx
```

Ex:
```
select  translate('welcome','em','x')  from dual;
wxlcox
```

9) REPLACE('str'/col, 'source string','target string')

Display given string by replacing source string with target string.

Ex:
```
select  replace('welcome','come','sys')  from dual;
welsys
```

Ex:
```
select  replace('welcome','come','X')  from dual;
welX
```

10) TRIM('str'/col)
Display given string by eleminating blank spaces before and after the string.

Ex:
```
select  trim('                    welcome to                ') " trim" || initcap('oracle')  from dual;
```

:welcome to Oracle

11) LTRIM('str'/col) [left trim]
Display given string by removing blank spaces from the left of string only.

Ex:
```
select  ltrim('                    welcome to                '), initcap('oracle')  from dual;
```

```
:welcome to                    Oracle
```

12) RTRIM('str'/col) [right trim]

Display given string by removing blank spaces from right of string only.

Ex:
```
select  rtrim('                    welcome to                 '), initcap('oracle')  from dual;

:                       welcome to Oracle
```

TRIM WITH KEYWORDS

13) LEADING 'ch' FROM 'str'/col

Display given string by removing similar occurrnaces of specific char from left of string

Ex:
```
select  trim(leading 'x' from 'xxxcxaxdxxxx')  from dual;
cxaxdxxxx
```

14) TRAILING 'ch' FROM 'str'/col

Display given string by removing similar occurances of specific char from right of string

Ex:
```
select  trim(trailing 'x' from 'xxxcaxdxxxx') from dual;
xxxcaxd
```

15) BOTH 'ch' FROM 'str'/col

Display given string by eleminating similar occurances of specific char from both sides of string.

Ex:
```
select  trim(both 'x' from 'xxxcaxdxxxx') from dual;
caxd
```

16) LPAD('str'/col,n,'ch') [left padding]

Display given string along with the specific char in the left of the string.

Ex:
```
select lpad('page 1',12,'*') from dual;

******page 1
```

17) RPAD('str'/col, n,'ch') [right padding]

Display given string along with specific char in the right of string.

Ex:
```
select  rpad('page 1',12,'*') from dual;

page 1******
```

DATE FUNCTIONS

1) **TO_DATE('char fmt of date',date)**

It will display any non-Oracle date format value in oracle's date format.

It accepts any char format of date(dd/mm/yy or dd-mm-yyyy or dd:mon:yyyy or yyyy-mm-dd) and converts it into oracle's default date format.

Ex:
```
select  to_date('22/03/2015','dd/mm/yyyy')  from dual;
22-mar-15
```

Ex:
```
select  to_date('2015:03:12','yyyy:mm:dd')  from dual;
12-mar-15
```

2) **ADD_MONTHS(d,n)**

Display a date value after adding " n " number of months to the specified date

Ex:
```
select  add_months(sysdate,6)  from dual;
```

Ex:
```
select  add_months(mfg,24) " exp "  from products;

        pid     pname   cost    mfg     exp
        --      ---     ---     --
        --      ---     ---     --
        --      --      ---     --
```

update the expdt for all products like 24 months from the date of mfg?

```
update prod_dtls set exp=add_months(mfg,24);
```

3) **MONTHS_BETWEEN(d1,d2)**

it shows number of months between dates.

Ex:
```
select months_between(sysdate,'21-may-13') from dual;
```

Ex:
```
select  months_between('01-jan-13','01-jan-14') from dual;
-12
```

4) **LAST_DAY(d)**

Display the date value of last day in the month.

Ex:
```
select  last_day('06-jul-10') from dual;
31-jul-10
```

5) NEXT_DAY(d,'weekdayname')

it will display the date value of given weekdayname after the specified date.

Ex: `select  next_day(sysdate,'saturday')  from dual;`

Ex: `select  next_day(sysdate,'monday')  from dual;`

CONVERSION FUNCTIONS

1) TO_NUMBER(chardata,[numberdata])

It accepts the chardata which contains a sequence of digits and convert it into number type data.

Ex:
```
salaries
--------
$1200
$11005              (char type data)
$107069.12
```

target: find the sum of salaries

```
1)  substr(sal,2)

        1200
        11005  ( chardata )
        107069.12

2)to_number(substr(sal,2))

converted into number type data

3) sum(to_number(substr(sal,2)))
```

Ex: `select sum(to_number(substr(sal,2)))  from emp;`

2) TO_CHAR(num, [char])
it accepts number type data and convert it into character type data.

Ex:
```
price_list                          target_format
---------                        ---------------
  100.78                            00,00,100.78
```

1200.12		00,01,200.12	
5463.00	(number type data)	00,05,463.00	(char format)
100700.00		01,00,700.00	
1223501.01		12,23,501.01	

Ex: select to_char(price_list,'00,00,000.00') from products;

3) TO_CHAR(date,[char])
It accepts Oracle's date type data and convert it into required char format.

Date Formats:

DD	digits of month
Day	Dayname
day	dayname
DAY	DAYNAME
Mon	3-chars of month with begining char capital
MON	"
mon	"
month	full month name(in lower case)
MONTH	full month name(caps)
Month	full month name with begining char capital
yy	last 2 digits of year
YY	"
yyyy	complete year number

Ex:

input	output
sysdate	27 September 2014

Ex: select to_char(sysdate,'dd month yyyy') from dual;

SAMPLE EXECUTIONS:

```
SQL> select  to_char(sysdate,'dd   month   yyyy')  from dual;

TO_CHAR(SYSDATE,'DDMO
---------------------
28   february    2015

SQL> select  to_char(sysdate,'dd   mon   yyyy')  from dual;

TO_CHAR(SYSDATE
---------------
28   feb   2015

SQL> select  to_char(sysdate,'dd   MON   yyyy')  from dual;

TO_CHAR(SYSDATE
---------------
28   FEB   2015
```

```
SQL> select  to_char(sysdate,'dd   Mon   yyyy')  from dual;

TO_CHAR(SYSDATE
---------------
28   Feb   2015

SQL> select  to_char(sysdate,'dd day  Mon   yyyy')  from dual;

TO_CHAR(SYSDATE,'DDDAYMO
------------------------
28 saturday   Feb   2015

SQL> select  to_char(sysdate,'dd Day  Mon   yyyy')  from dual;

TO_CHAR(SYSDATE,'DDDAYMO
------------------------
28 Saturday   Feb   2015

SQL> select  to_char(sysdate,'dd DAY  Mon   yyyy')  from dual;

TO_CHAR(SYSDATE,'DDDAYMO
------------------------
28 SATURDAY   Feb   2015
```

PL/SQL

PL/SQL is a procedural extension to Oracle SQL. PL/SQL is integrated with Oracle Database, enabling you to use all of the Oracle Database SQL statements, functions, and data types. You can use PL/SQL to control the flow of a SQL program, use variables, and write error-handling procedures.

A primary benefit of PL/SQL is the ability to store application logic in the database itself. A **PL/SQL procedure** or **function** is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or to perform a set of related tasks. The principal benefit of server-side programming is that built-in functionality can be deployed anywhere.

Oracle Database can also store program units written in Java. A Java stored procedure is a Java method published to SQL and stored in the database for general use. You can call existing PL/SQL programs from Java and Java programs from PL/SQL.

It is a procedural language using sql concepts.
It is an extension to the sql.

pl/sql program

It is a collection of programming stmts and with sql concepts (queries) to perform user required tasks .

What is pl/sql?
Pl/sql is a collection of User defined objects like Programs, procedures, Functions, Triggers, Types, Packages and so on.

Advantages:

- Multiple queries are executed parllelly.
- It reduces number of hits to the database.
- In pl/sql user can create the objects according to his requirements.
- It increases n/w performance.
- Modularity (dividing a big task into smaller modules)
- Enhansibility (It can easily accept the future changes)
- Reusability

2 Categories of Programs
i) Anonymous Blocks (Programs)
ii) Sub Programs   (Stored Procedures and Functions)

i) Anonymous Blocks

It is a program which is not saved inside the DataBase.

2 types of anonymous blocks

a) Static Programs
It is not accepting runtime input values and Always generate same output, Input value is fixed.

b) **Dynamic Programs**
It will accept RunTime Input values and Generates different outputs based on different input values.

STRUCTURE OF PL/SQL PROGRAM:

```
DECLARE             [ Optional ]
<declaration stmts >;
BEGIN               [Mandatory ]
<Assignment stmt>;
<Output stmts>;
<Data processing stmts>;
EXCEPTION           [Optional ]
<Error handling stmts>;
END;                 /* end of program */
```

<u>DECLARE block</u>

It contains declaration stmts. Used to declare variables. So these variables get memory space from the database engine based on their datatype and size.

Syntax:

```
var_name       DATATYPE(size);
```

Ex:

```
v_eno  int;
v_name          varchar2(20);
v_sal   number(5);
v_jdatedate;
```

<u>BEGIN block</u>

Assignment statements

Used to store values into the declared variables by using assignment operator [:=] or by using SELECT stmt   with INTO keyword.

Syntax-1:
BY USING ***ASSIGNMENT*** OPERATOR

```
var_name:=value / expression;
```

Ex:

```
v_eno:=7654;
```

Ex:
```
declare
x	int;
y	int;
z	int;
begin
x:=100;
y:=200;
z:=x+y;
```

Syntax-2:
By Using **SELECT Query with INTO keyword**

```
select col1, col2, ....,coln INTO var_name1, varname_2,....var-n
from table
where <cond>;
```

Ex:
```
select ename,sal,hiredate INTO V_name,v_sal,v_jdate
from emp where empno=v_eno;
```

Out Put stmts

Used to display values of variables and normal messages.
Oracle provides a predefined output function as follows.

DBMS_OUTPUT.PUT_LINE ('normal messages' or var_name);

Ex:
```
dbms_output.put_LINE(' employee information ');
dbms_output.put_LINE('--------------------------');
dbms_output.put_line(v_eno);
dbms_output.put_line(v_name);
dbms_output.put_line(v_sal);
dbms_output.put_line(v_jdate);
```

Data Processing stmts
Any sql query is known as data processing stmt.

EXCEPTION Block
It is used to display user friendly error messages instead of system generated error messages.

END;
Indicates end of program.

HOW TO "COMPILE" AND "EXECUTE" THE PL/SQL PROGRAM IN sql * plus WINDOW?

/	Used to compile and execute the pl/sql program in SQLPLUS environment.
--	It is a single line comment symbol

/* text */ It is a multi line comment symbol

NOTE:
By default any program or procedure should not display output. To display output,

SET SERVEROUTPUT ON;
It enable oracle engine to display output from any pl/sql program or procedure.
This is valid for current session.

Ex: write a program to display welcome message .

```
begin
dbms_output.put_line ('welcome to oracle pl/sql');
end;
```

Ex: write a program to display addition , average, max and min of 3000 and 5000?

```
DECLARE
X       INT;
Y       INT:=5000;
S       INT;
A       NUMBER(7,2);
MX      INT;
MN      INT;
BEGIN
X:=3000;
S:=X+Y;
A:=S/2;
SELECT GREATEST(X,Y) INTO MX FROM DUAL;
SELECT LEAST(X,Y) INTO MN       FROM DUAL;
DBMS_OUTPUT.PUT_LINE(' SUM=');
DBMS_OUTPUT.PUT_LINE(S);
DBMS_OUTPUT.PUT_LINE('AVERAGE=');
DBMS_OUTPUT.PUT_LINE(A);
DBMS_OUTPUT.PUT_LINE('HIGHER VALUE=');
DBMS_OUTPUT.PUT_LINE(MX);
DBMS_OUTPUT.PUT_LINE('Least VALUE=');
DBMS_OUTPUT.PUT_LINE(MN);
END;
/
```

sample execution:

```
SUM=
8000
AVERAGE=
4000
HIGHER VALUE=
5000
MIN VALUE=
3000
```

PL/SQL procedure successfully completed.

NOTE:

How can i display a normal mesg after that the value of variable immediately in the same line?

By using concatenation Operator [||]
It will display the value or message after the previous message or value.

```
DECLARE
        X       INT;
        Y       INT:=5000;
        S       INT;
        A       NUMBER(7,2);
        MX      INT;
        MN      INT;
        BEGIN
        X:=3000;
        S:=X+Y;
        A:=S/2;
        SELECT GREATEST(X,Y) INTO MX FROM DUAL;
        SELECT LEAST(X,Y) INTO MN       FROM DUAL;
        DBMS_OUTPUT.PUT_LINE(' SUM='||s);
        --DBMS_OUTPUT.PUT_LINE(S);
        DBMS_OUTPUT.PUT_LINE('AVERAGE='||a);
        --DBMS_OUTPUT.PUT_LINE(A);
        DBMS_OUTPUT.PUT_LINE('HIGHER VALUE='||mx);
        --DBMS_OUTPUT.PUT_LINE(MX);
        DBMS_OUTPUT.PUT_LINE('MIN VALUE='||mn);
        --DBMS_OUTPUT.PUT_LINE(MN);
        END;
        /
```

sample execution:

```
anonymous block completed
 SUM=8000
AVERAGE=4000
HIGHER VALUE=5000
MIN VALUE=3000
```

Types of pl/sql programs: 2

1) Static programs
Always generate same output Since the input value in the program is fixed and constant.

2) Dynamic programs
Generates different output values based on different input values. In this case the program can accept runtime input values.

STATIC PROGRAMS

Ex: write a program to display the employee information of empID 7654?

```
declare
veno    int:=7654;
vname varchar2(20);
vsal    number(5);
vjob    varchar2(20);
vhiredate       date;
vcommnumber(4);
vdeptno         number(3);
BEGIN
select ename,sal,job,hiredate,comm,deptno INTO
        vname,vsal,vjob,vhiredate,vcomm,vdeptno
from emp
where empno=veno;
dbms_output.put_line(' Info of 7654');
dbms_output.put_line('---------------');
dbms_output.put_line(vname);
dbms_output.put_line(vsal);
dbms_output.put_line(vjob);
dbms_output.put_line(vhiredate);
dbms_output.put_line(vcomm);
dbms_output.put_line(vdeptno);
END;
/
```

sample output:

```
Info of 7654
MARTIN
1250
SALESMAN
28-SEP-81
1400
30

PL/SQL procedure successfully completed.
```

Ex:

```
declare
veno    int:=7654;
vname varchar2(20);
vsal    number(5);
vjob    varchar2(20);
vhiredate       date;
vcommnumber(4);
vdeptno         number(3);
BEGIN
select ename,sal,job,hiredate,comm,deptno INTO
        vname,vsal,vjob,vhiredate,vcomm,vdeptno
from emp
where empno=veno;
dbms_output.put_line(' Info of Emp id:       7654');
```

```
        dbms_output.put_line('===============================================');
        dbms_output.put_line('Name of emp: '||vname);
        dbms_output.put_line('Salary of emp: '||vsal);
        dbms_output.put_line('Designition of emp: '||vjob);
        dbms_output.put_line('Join date of emp: '||vhiredate);
        dbms_output.put_line('Commission of emp: '||vcomm);
        dbms_output.put_line('Deptno of emp: '||vdeptno);
        dbms_output.put_line('===============================================');
        END;
        /
```

output:

```
Info of 7654
===============================================
Name of emp: MARTIN
Salary of emp: 1250
Designition of emp: SALESMAN
Join date of emp: 28-SEP-81
Commission of emp: 1400
Deptno of emp: 30
===============================================

PL/SQL procedure successfully completed.
```

Ex: Write a program to display number of customers from the city " Delhi " ?

```
DECLARE
VCITY   VARCHAR2(20):='Delhi';
CUST_CNT  INT;
BEGIN
SELECT COUNT(*) INTO CUST_CNT FROM CUST_DTLS WHERE CITY=VCITY;
DBMS_OUTPUT.PUT_LINE(' NUMBER OF CUSTOMERS FROM '||VCITY||' =
'||CUST_CNT);
END;
```

Ex:
Write a program to display number of male customers and number of female customers?

```
        declare
        male_cnt        int;
        female_cnt      int;
        begin
        select count(*) into male_cnt from cust_dtls where gender='M';
        select count(*) into female_cnt from cust_dtls where gender='F';
        dbms_output.put_line(' Number of males= '||male_cnt);
        dbms_output.put_line(' Number of Females='||female_cnt);
        end;
```

Assignments:

1) Write a program to display designition of empid 7788?

2) Write a program to display the city and mobile number of customer id " cust-5"?

3) Write a program to display the location of department " RESEARCH "?

Dynamic Programs

By using **& (Address Of)** operator we will make a program as a dynamic program. Instead of assigning a constant value in to a variable , While the execution of the program, the program has to take a value from the end user and that value stored it into a variable.

varname:=&varname;

Ex: write a program to display the details of employee for the given empno?

```
        declare
        v_eno  number(4);
        v_ename         varchar2(20);
        v_sal   number(5);
        v_job   varchar2(20);
        v_hiredate      date;
        v_comm          int;
        v_mgr_code int;
        v_deptno  int;
        BEGIN
        v_eno:='&v_eno';
        select ename,sal,job,hiredate,comm,mgr,deptno INTO
v_ename,v_sal,v_job,v_hiredate,v_comm,v_mgr_code,
v_deptno
        from emp
        where empno=v_eno;
        dbms_output.put_line(v_eno||' Employee  information ');
        dbms_output.put_line
('====================================================');
        dbms_output.put_line
('EMPNAME  : EMP-SALARY  :       EMP-DESG  : EMP-JOINDATE  : comm    :   manager
code   :   deptno');
        dbms_output.put_line
('====================================================');
        dbms_output.put_line
(v_ename||'                '||v_sal||'        '||v_job||'         '||v_hiredate||'       '||v_comm||'
        '||v_mgr_code||'         '||v_deptno);
        dbms_output.put_line
('====================================================');
        end;
```

Ex: using %rowtype

```
        declare
        v_eno  number(4);
        r  emp%rowtype;
        BEGIN
        v_eno:='&v_eno';
        select * INTO r         from emp
        where empno=v_eno;
        dbms_output.put_line(v_eno||' Employee  information ');
        dbms_output.put_line
('==================================================');


        dbms_output.put_line(r.ename||'  '||r.sal||'  '||r.job||'  '||r.mgr||'  '||r.hiredate||'
'||r.comm||'  '||r.deptno);
        dbms_output.put_line
('==================================================');
        end;
```

output:

```
7654  information
==================================================
MARTIN              1250    SALESMAN    28-SEP-81
==================================================
```

output:

```
7788  information
==================================================
SCOTT               3000    ANALYST     19-APR-87
==================================================
```

Ex: Write a program to display the number of emps in the given deptno?

```
        declare
        vdno    number(2);
        e_count         int;
        begin
        vdno:=&vdno;
        select count(empno) into e_count from emp where deptno=vdno;
        dbms_output.put_line(' number of emps in Department : '||vdno||' :  = '||e_count);
        end;
```

output:

```
 number of emps in 20 = 5
```

Ex:

```
number of emps in 30 = 6
```

Ex: write a program to display the dept details under which the employee is working with given id?

```
declare
veno    number(4);
vdno    number(4);
vdname          varchar2(20);
vloc     varchar2(20);
begin
veno:=&veno;
select * into vdno,vdname,vloc from dept
where deptno=(select deptno from emp where empno=veno);
dbms_output.put_line(' department details of employee : '||veno);
dbms_output.put_line(vdno||' '||vdname||'      '||vloc);
end;
```

Ex output:

```
 department details of employee : 7902
20      RESEARCH    DALLAS
```

Ex output:

```
department details of employee : 7788
20      RESEARCH    DALLAS
```

Assignment

1) write a program to display the total salary i am paying to deptno 30 employees?

2) write a program to display the "number of male customers" from the given city?

3) Write a program to display the number of emps working under given deptno?

4) write a program to find the number of emps working with given designition?

5) WRITE A PROGRAM TO DISPLAY NUMBER OF PRODUCTS HAVING WARRENTY MANUFACTURED IN THE YEAR 2014?

```
DECLARE
nop  int;
begin
select count(*) into nop from prod_dtls
where mfg between '01-jan-14' and '31-dec-14'
          and
          warrenty is not  null;
dbms_output.put_line(' NUmber of products having warrenty='|| nop);
end;
```

Ex
write a program to display customer account details for the given customer name?

```
declare
vcname  varchar2(20);
vactno  number(12);
vacttype  varchar2(10);
vopendate date;
vactbal  number;
vccode  varchar2(10);
begin
vcname:='&vcname';
select actno,act_type,act_open_date,act_bal,cust_code into
      vactno,vacttype,vopendate,vactbal,vccode
      from cust_act_dtls
      where cust_code =(select cno from cust_dtls where cname=vcname);

dbms_output.put_line('Account details of  '||upper(vcname));
dbms_output.put_line('************************************');
dbms_output.put_line('Account Number='||vactno);
dbms_output.put_line('Account Type='||vacttype);
dbms_output.put_line('Account Open Date='||vopendate);
dbms_output.put_line('Account Balance='||vactbal);
dbms_output.put_line('Customer Code='||vccode);
end;
```

Note:

Generally, we need to declare a variable with column data type otherwise we will get compatibility issues.

To store a complete record in to a variable we need declare a variable as TABLE BASED RECORD type variable.

TYPE COMPATIBILITY KEYWORDS

We can declare the variables dynamically/ Implicitly as column DATA type (using **%TYPE**) and record type (by using **%ROWTYPE**).
This eliminates Data type and size incompatibility issues.

To Declare variables implicitly

%TYPE

It is used to declare a variable of column data type dynamically.

Syntax: var_name   table_name.col_name%type;

Ex: vdno   emp.deptno%type;

In the above the %TYPE is taking the data type and size of the column DEPTNO from the table EMP and these things will be applicable to the varaible.

Ex: vacctno   cust_act_dtls.actno%type;

PL/SQL RECORDS: 3 types

i) Table based records   %ROWTYPE
This record type variable supports all columns from the table

ii) Cursor based records   %rowtype
This record type variable supports all columns from the cursor

iii) User defined Records   TYPE with keyword IS RECORD(......);
This record type variable supports all columns from the user defined record structure.

%ROWTYPE

Used to declare a Table Based RECORD TYPE variable, which will supports all columns from that table.

syn:

var_name   tablename%ROWTYPE;

Ex: emp_rec   emp%rowtype;

Ex: prod_info   prod_dtls%rowtype;

```
                         emp_rec
                            |
----------------------------------------------------------------------------
|      |          |          |          |          |      |
empno ename       sal        job        hiredate   comm  deptno
```

Ex: select * into emp_rec from emp where empno=7788;

How to access individual values from record type variable?

Syntax:

record_var_name . colname;

Ex: dbms_output.put_line(' emp salary :'|| emp_rec.sal);

Ex: dbms_output.put_line(' emp Info :'|| emp_rec);----------->Don't specify like this

---

Ex:
Write a program to display the information of employee for the given employee id?

```
declare
veno    emp.empno%type; --Dynamic declaration of variable
e_rec   emp%rowtype;           --Declaring table based record type variable
begin
veno:=&veno;
select * into e_rec from emp where empno=veno;
dbms_output.put_line(' given employee id: '||veno);
dbms_output.put_line(' Information of  '||veno);
dbms_output.put_line(e_rec.ename||' '||e_rec.sal||' '||e_rec.job||' '||e_rec.hiredate||'
'||e_rec.comm||' '||e_rec.deptno);
end;
```

Ex:
write a program to display the information of product for the given product id?

```
declare
vpid     prod_dtls.pid%type;
p_rec    prod_dtls%rowtype;

begin
vpid:='&vpid';
select * into p_rec from prod_dtls where pid=vpid;
dbms_output.put_line(' information of prodid: '||vpid);
dbms_output.put_line(p_rec.pname||'          '||p_rec.cost||'  '||p_rec.mfg||'
'||p_rec.warrenty||' '||p_rec.prod_comp_id);
end;
```

Ex
write a program to display the information of a customer for the given customer number?

```
declare
vcustid cust_dtls.custid%type;
cust_rec          cust_dtls%rowtype;
begin
vcustid:='&vcustid';
select * into cust_rec from cust_dtls where custid=vcustid;
dbms_output.put_line(' given customer id: '||vcustid);
```

```
        dbms_output.put_line(' customer name='||cust_rec.custname);
        dbms_output.put_line(' customer city='||cust_rec.custcity);
        dbms_output.put_line(' customer gender='||cust_rec.custgender);
        dbms_output.put_line(' customer mobile number='||cust_rec.custmobile);
        end;
```

```
declare
        vcustid customers_br1.cid%type;
        cust_rec        customers_br1%rowtype;
        begin
        vcustid:='&vcustid';
        select * into cust_rec from customers_br1 where cid=vcustid;
/*      dbms_output.put_line(' given customer id: '||vcustid);
        dbms_output.put_line(' customer name='||cust_rec.cname);
        dbms_output.put_line(' customer city='||cust_rec.city);
        dbms_output.put_line(' customer gender='||cust_rec.gender);
        dbms_output.put_line(' customer mobile number='||cust_rec.mobile); */
  dbms_output.put_line('GIVEN ID:'||vcustid);
  dbms_output.put_line('-----------------------------------------------------------');
  dbms_output.put_line('NAME       CITY        GENDER        MOBILE-NUMBER'       );
  dbms_output.put_line('------------------------------------------------------------');
  dbms_output.put_line(cust_rec.cname||'     '||cust_rec.city||'      '||cust_rec.gender||'
'||cust_rec.mobile);
        end;
```

SUB PROGRAMS

PROCEDURES
FUNCTIONS

STORED PROCEDURES or PL/SQL PROCEDURES

A named program which is stored / created under the data base is known stored procedure.

It is also known as a subprogram to perform a task or set of tasks.

Features of a Procedure:-

- It can be stored as a precompiled object.
- Means it will be compiled once and executed any number of times.
- Procedures provides reusability.
- Procedures can be easily enhansible for future requirements.
- Procedures are EXPLICITLY executed by the user.
- Procedures are executed by using the command EXECUTE / EXEC.
- Procedures may or may not take arguments
- Procedures may called from other procedures or functions
- By Default,Procedure cannot return any value to the calling object(Program or procedure or function).

Procedures are 2 types

a) Static Procedure
It will not contains any argument variables and it wont take any values from the user at runtime.
Always display same output.

b) Dynamic procedure
A procedure with argument variables is known as dynamic procedure.
It will take runtime input values from the user and display different output values for different users.

PROCEDURE CONTAINS 2 PARTS

i) Procedure specification
It contains creation of procedure information like proc_name and argument list

ii) Procedure body
It contains the code for the functionality of procedure.

Syntax:

```
            /*Procedure specification*/

      create [or replace]  PROCEDURE <proc_name> [(arg_var1  datatype, arg_var2
datatype,......)]
      IS / AS

      /* Procedure body */

      <declaration stmts>;
      BEGIN
      <assignment stmts>;
      <data processing stmts>;
      <output stmts>;
      EXCEPTION
      <error handling stmts>;
      END <proc_name>;
      / ( to compile the procedure in SQL * PLUS window )

      Procedure created.
```

HOW TO EXECUTE THE PROCEDURE?

EXECUTE <proc_name>[(arg_val1, arg_val2,.....)];

STATIC PROCEDURE

Ex: write a procedure which is finding number of emps working under dept 30?

```
      create  procedure proc_ecount_30
```

```
is
e_count         int;
begin
select count(*) into e_count from emp where deptno=30;
dbms_output.put_line(' Number of emps under deptno 30 are '||e_count);
end proc_ecount_30;
```

Ex: `execute proc_ecount_30;  /* Executing procedure explicitly */`

How do i execute a procedure implicitly?
By making a call to the procedure we can implicitly execute the procedure.

--Calling a procedure from any program

```
BEGIN
proc_ecount_30; --Procedure calling stmt
dbms_output.put_line(' calling procedure is finished');
dbms_output.put_line(' Program Execution is finished');
end;
```

DYNAMIC PROCEDURES

--WRITING THE PROCEDURE TO DISPLAY THE NUMBER OF EMPS IN THE GIVEN DEPTNO?

```
CREATE PROCEDURE PROC_ecount_did(VDNO  EMP.DEPTNO%TYPE)
IS
CNT INT;
BEGIN
SELECT COUNT(*) INTO CNT FROM EMP WHERE DEPTNO=VDNO;
DBMS_OUTPUT.PUT_LINE(' NUMBER OF EMPS IN DEPT '||VDNO||' ARE : '||CNT);
END PROC_ecount_did;
```

SAMPLE EXECUTIONS:

```
EXEC PROC_2(10);
EXEC PROC_2(20);
EXEC PROC_2(30);
```

```
anonymous block completed
 NUMBER OF EMPS IN DEPT 10 ARE  2

anonymous block completed
 NUMBER OF EMPS IN DEPT 20 ARE  5

anonymous block completed
 NUMBER OF EMPS IN DEPT 30 ARE  6
```

ex:

```
declare
vdno int;
begin
vdno:=&vdno;
proc_ecount_did(vdno); --Procedure calling statement
dbms_output.put_line(' Exec Finished');
end;
```

WRITING PROCEDURE CALLING STATEMENTS IN A PROGRAM:-

For the following program we need to create 2 procedures called p_add and p_multi.

```
declare
x int;
y int;
begin
x:=&x;
y:=&y;
dbms_output.put_line(' adding '||x ||' and '||y);
Dbms_output.put_line(' Making a call to P_ADD procedure');
p_add(x,y);
dbms_output.put_line(' Multiplication of '||x||' and '||y);
dbms_output.put_line(' make a call to P_MULTI procedure');
p_multi(x,y);
dbms_output.put_line(' 2 tasks successfully completed');
end;
```

STATIC PROCEDURE:

Ex: write a procedure to display the customer details and his account details for the custid "cust-5"?

```
        create or replace procedure proc_CUST_ACT_DTLS
        is
        v_cname                 cust_dtls.cname%type;
        v_gender        cust_dtls.gender%type;
        v_city          cust_dtls.city%type;
        v_actno                 cust_act_dtls.actno%type;
        v_act_type      cust_act_dtls.act_type%type;
        v_act_bal       cust_act_dtls.act_bal%type;
        begin
        select c.cname,c.gender,c.city,ca.actno,ca.act_type,ca.act_bal INTO
             V_CNAME,v_gender,v_city,v_actno,v_act_type,v_act_bal
        from cust_dtls c , cust_act_dtls ca
        where c.cno='cust-5'
                and
                c.cno=ca.cust_code;
        dbms_output.put_line(' Details of  customer 5');
        dbms_output.put_line(v_cname||' '||v_gender||' '||v_city||' '||v_actno||' '||v_act_type||'
'||v_act_bal);
        end proc_CUST_ACT_DTLS;
```

DYNAMIC PROCEDURE:

Ex: write a procedure to display the customer details and his account details for the GIVEN custid ?

```
create or replace procedure proc_3(VCID     cust_dtls.cno%type)
is
v_cname                    cust_dtls.cname%type;
v_gender         cust_dtls.gender%type;
v_city           cust_dtls.city%type;
v_actno                    cust_act_dtls.actno%type;
v_act_type       cust_act_dtls.act_type%type;
v_act_bal        cust_act_dtls.act_bal%type;
begin
select c.cname,c.gender,c.city,ca.actno,ca.act_type,ca.act_bal INTO
     V_CNAME,v_gender,v_city,v_actno,v_act_type,v_act_bal
from cust_dtls c , cust_act_dtls ca
where c.cno=VCID
          and
          c.cno=ca.cust_code;
dbms_output.put_line(' Details of customer '||VCID);
dbms_output.put_line(v_cname||' '||v_gender||' '||v_city||' '||v_actno||' '||v_act_type||'
'||v_act_bal);
end proc_3;
```

STATIC PROCEDURE:

Ex: write a procedure to display the employee details and dept details for the empno 7902?

```
create or replace procedure proc_4
is
v_ename          emp.ename%type;
v_job   emp.job%type;
v_sal   emp.sal%type;
v_hiredate emp.hiredate%type;
v_dno  emp.deptno%type;
v_dname          dept.dname%type;
v_loc   dept.loc%type;
begin
select e.ename,e.sal,e.job,e.hiredate,e.deptno,d.dname,d.loc INTO
          v_ename,v_sal,v_job,v_hiredate,v_dno,v_dname,v_loc
from emp e , dept d
where e.empno=7902  and e.deptno=d.deptno;
dbms_output.put_line('Details of empno= 7902');
dbms_output.put_line(v_ename||' '||v_job||' '||v_sal||' '||v_hiredate||' '||v_dno||'
'||v_dname||' '||v_loc);
end proc_4;
```

DYNAMIC PROCEDURES:

EX: write a procedure to display the details of employee for the given employee id?

```
        create or replace procedure proc_5 ( vempid emp.empno%type)
        is
        emp_rec         emp%rowtype;
        begin
        select * into emp_rec from emp where empno=vempid;
        dbms_output.put_line(' employee details of '||vempid);
        dbms_output.put_line ('************************************************');
        dbms_output.put_line  ( emp_rec.ename||','||emp_rec.sal||','
                                                        ||emp_rec.job||','||emp_rec.hiredate);
        end proc_5;
```

sample output:

```
 employee details of 7902
FORD,3000,ANALYST,03-DEC-81

anonymous block completed
 employee details of 7788
SCOTT,3000,ANALYST,19-APR-87

anonymous block completed
 employee details of 7654
MARTIN,1250,SALESMAN,28-SEP-81
```

Ex:	write a procedure to display the customer mobile number and name of the given custid if he is male and from the	city delhi?

```
desc cust_dtls
Name       Null Type
---------- ---- ------------
CUSTID          CHAR(4)
CUSTNAME        VARCHAR2(20)
CUSTCITY        VARCHAR2(10)
CUSTGENDER      CHAR(1)
CUSTMOBILE      NUMBER(10)

create or replace procedure proc_5(vcid       cust_dtls.cno%type) is
        --vmobile                  cust_dtls.mobile%type;
        vcustname      cust_dtls.cname%type;
        begin
        select cname into vcustname from cust_dtls where cno=vcid and gender='M' and
city='Delhi';
        dbms_output.put_line(' give customer id='||vcid);
        dbms_output.put_line(' he is male and he is from delhi');
        --dbms_output.put_line('mobile number is -'||vmobile);
        dbms_output.put_line(' His  name is  '||vcustname);
```

```
end proc_5;
```

Ex: write a procedure to display number of emps working under given department name?

Ex: write a procedure to display the number of customers from the given city with given gender?

Ex: write a procedure to display the number of accounts of given account type?

PL/SQL RECORDS OR COLLECTIONS

Generally in plsql we have 2 types of variables.
They are Scalar Variables and Composite Variables.

Scalar variable
It is able to store one value at a time and that value with any one data type.

Ex:
```
veno    int;
vactno  act_dtls.actno%type;
```

Composite variable
This variable is able to store collection of values with different data types and also with same datatype.
That's why it is known as a pl/sql RECORD or COLLECTION.

types of RECORD type variables

a) Table based record
b) Cursor based record
c) User defined record
d) Multidimensional array type record or pl sql table
e) Single dimensional array type record.

HOW DO I CREATE A RECORD OR COLLECTION:-

By using 3-methods. (3 types of plsql records)

i) Table based records
It supports all columns from the table.

```
syn:    varname             tablename%ROWTYPE;
```

ii) Cursor Based Records
It supports all columns from the cursor.

```
syn:    varname             cursorname%ROWTYPE;
```

iii) User Defined records:
A record is a collection of columns.

A table based record is a collection of all columns from the table.
A user defined record is a collection of columns specified in the record structure.
It supports the columns specified in the user defined structure.
User can define his required record strucutre by using the keyword "TYPE".

SYN: creating user defined record structure

```
TYPE  <typename>  IS RECORD(
                              col1    datatype(size),
                              col2    datatype(size),
                              :       :       :
                              :               :
                              );
```

Ex:

```
TYPE  emprec  IS RECORD(
                              empname       emp.ename%type,
                              esal    emp.sal%type,
                              ejob    emp.job%type

                              );
```

SYNTAX: Declaring variables as user defined record type.

```
varname       <typename>;
```

Ex: r emprec; /* Here Oracle engine reserves only 3 column length memory */

erec emp%rowtype; /* Here oracle engine reserves the memory required for all columns from the table emp */

Ex:
Write a procedure to display the name,sal,job,hiredate,deptno for the given employee id?

```
create or replace procedure p_emp( veno emp.empno%type)
is
type emprec is record(
                    name  emp.ename%type,
                    esal  emp.sal%type,
                    desg  emp.job%type,
                    jdate emp.hiredate%type,
                    dno   emp.deptno%type
                    );

rec   emprec;
begin
```

```
select ename,sal,job,hiredate,deptno into rec
from emp where empno=veno;
dbms_output.put_line(' Ename='||rec.name);
dbms_output.put_line(' EmpSal='||rec.esal);
dbms_output.put_line(' Emp Job Title='||rec.desg);
dbms_output.put_line(' Emp join date='||rec.jdate);
dbms_output.put_line(' Emp Dept No='||rec.dno);
end p_emp;
```

Advantages:

i) It is reducing the number of declarations of variables.
ii) It is reducing the amount of memory reserved for Record based variable.

Ex: Write a procedure to display the name,sal,job,dno,dname,loc for the given employee id?

```
create or replace procedure p_emp_dept( veno emp.empno%type)
is
type edrec is record(
                        name  emp.ename%type,
                        esal  emp.sal%type,
                        desg  emp.job%type,
                        dno   emp.deptno%type,
                        dname dept.dname%type,
                        city  dept.loc%type
                        );

r   edrec;              /* User Defined Record Based Variable */
begin
select e.ename,e.sal,e.job,e.deptno,d.dname,d.loc into r
from emp e , dept d where e.empno=veno and e.deptno=d.deptno;
dbms_output.put_line(' Ename='||r.name);
dbms_output.put_line(' EmpSal='||r.esal);
dbms_output.put_line(' Emp Job Title='||r.desg);
dbms_output.put_line(' Emp Dept No='||r.dno);
dbms_output.put_line(' Emp dept name='||r.dname);
dbms_output.put_line(' Emp dept city='||r.city);
end p_emp_dept;
```

Ex: Write a procedure to display the name,cost and warrenty of the product for the given product id?

Ex: Write a procedure to display pname,cost,mfg,exp,companyName,Dealer address,phonenumber
for the given product id?

Ex: write a procedure to display customer name, actno,act balance and act name for the given customer id?

```
create or replace procedure p_cust_act_info( vcid cust_dtls.cno%type)
```

```
        is
        type custrec is record(
                                cname  cust_dtls.cname%type,
                                actno  cust_act_dtls.actno%type,
                                actname  act_types.act_name%type,
                                bal      cust_act_dtls.act_bal%type
                                                        );

        rec   custrec;
        begin
        select cd.cname,cad.actno,at.act_name,cad.act_bal INTO rec
        from cust_dtls cd Inner join cust_act_dtls cad
  ON cd.cno=cad.cust_code
         Inner Join act_types at
         ON cad.act_type=at.act_type
  WHERE cd.cno=vcid;

        dbms_output.put_line(' Cust name='||rec.cname);
        dbms_output.put_line(' Cust Act No='||rec.actno);
        dbms_output.put_line(' Cust Act Name='||rec.actname);
        dbms_output.put_line(' Account Bal='||rec.bal);
              end p_cust_act_info;
```

CONDITIONAL STATEMENTS

These are used to execute a set of statements based on condition result.
The conditional statements are used to compare values.

1) simple if

It is having only true part. If the condition is true the it executes the stmts.

```
Syntax:        if (condition) then
        stmt;
        stmt;
        :
        end if;
```

2) if -- then-- else

It is having both true and false parts. If the condition is true then it executes the true part and if the condition was failed then it executes the false part.

```
Syntax:        if (condition) then
        stmt;
        stmt;
        :
        ELSE
        stmt;
```

```
stmt;
:
end if;
```

3) COMPUND IF

checking if condition with in other if condition.

```
Syntax:     if (cond-1) then
        stmts;
        stmts;
            if (cond-2) then
            stmt;
            stmts;
            :
            else
            stmt;
            :
            end if; --closing of cond-2
        else
        stmt;
        :
        end if; --closing of cond-1
```

4) ELSIF construct

checking if condition with in else part of other if.

```
Syntax:     if (cond) then
        stmts;
        :
        else
        if ( cond) then
        stmts;
        :
          else
            stmts;
            :
          end if
        stmts;
        stmts;
        :

        end if;

            or

        if (cond) then
        stmt;
        stmt;
        :
        ELSIF(cond) then
            stmt;
            stmt;
            else
            stmts;
```

```
                :
         END IF;
```

Ex:
Creating a procedure to display biggest of 2 integers with simple if logic?

```
CREATE OR REPLACE PROCEDURE PROC_BIGINT_1( X INT, Y INT)
IS
BEGIN
DBMS_OUTPUT.PUT_LINE('X='||X||'  '||'Y='||Y);
IF (X>Y) THEN

  DBMS_OUTPUT.PUT_LINE('FIRST VALUE IS BIG');
  END IF;
    DBMS_OUTPUT.PUT_LINE('end of execution');
  END PROC_BIGINT_1;

  EXEC PROC_BIGINT_1(10,20);
  EXEC PROC_BIGINT_1(100,20);
```

Ex:
Creating a procedure to display biggest of 2 integers with if-then-else logic?

```
CREATE OR REPLACE PROCEDURE PROC_BIGINT_2( X INT, Y INT)
IS
BEGIN
DBMS_OUTPUT.PUT_LINE('X='||X||'  '||'Y='||Y);
IF (X>Y) THEN
   DBMS_OUTPUT.PUT_LINE('FIRST VALUE IS BIG');
  else
  DBMS_OUTPUT.PUT_LINE('Second VALUE IS BIG');
  END IF;
    DBMS_OUTPUT.PUT_LINE('end of execution');
  END PROC_BIGINT_2;

  exec proc_bigint_2(20,10);
  exec proc_bigint_2(100,100);
```

Ex:
Creating a procedure to display biggest of 2 integers with Elsif logic?

```
CREATE OR REPLACE PROCEDURE PROC_BIGINT_3( X INT, Y INT)
IS
BEGIN
DBMS_OUTPUT.PUT_LINE('X='||X||'  '||'Y='||Y);
IF (X>Y) THEN
   DBMS_OUTPUT.PUT_LINE('FIRST VALUE IS BIG');
```

```
  elsif(x=y) then
    dbms_output.put_line(' x and y are equal');
     else
  DBMS_OUTPUT.PUT_LINE('Second VALUE IS BIG');
  END IF;
     DBMS_OUTPUT.PUT_LINE('end of execution');
  END PROC_BIGINT_3;

  exec proc_bigint_3(4,5);
  exec proc_bigint_3(5,5);
  exec proc_bigint_3(14,5);
```

Ex:
Write a program to display biggest of integers if any one given value is null?

```
CREATE OR REPLACE PROCEDURE PROC_BIGINT_4( X INT, Y INT)
IS
a int;
b int;
BEGIN
DBMS_OUTPUT.PUT_LINE('X='||X||'  '||'Y='||Y);
if x is null then
dbms_output.put_line(' Enter a valid value in the First position');
else
a:=x;
end if;
if y is null then
dbms_output.put_line(' Enter a valid value in the second position');
else
b:=y;
end if;

IF (a>b) THEN
    DBMS_OUTPUT.PUT_LINE('FIRST VALUE IS BIG');
  elsif(a=b) then
    dbms_output.put_line(' Values are equal');
     else
  DBMS_OUTPUT.PUT_LINE('Second VALUE IS BIG');
  END IF;
     DBMS_OUTPUT.PUT_LINE('end of execution');
  END PROC_BIGINT_4;
```

Ex:
write a procedure to find which type of customers(either male customers or female customers) less in count,   from the given city?

```
create or replace procedure proc_Gender_cnt
(vcity cust_dtls.city%type) is
```

```
v_mcntint;
v_fcnt  int;
begin
select count(*) into v_mcnt
 from cust_dtls where city=vcity and gender='M';
select count(*) into v_fcnt
from cust_dtls where city=vcity and gender='F';
dbms_output.put_line('Given city Name:- '||vcity);
dbms_output.put_line('Male count:- '||v_mcnt);
dbms_output.put_line('Female count:- '||v_fcnt);
if(v_mcnt < v_fcnt) then
dbms_output.put_line
('Males are less in count than females');
elsif(v_mcnt=v_fcnt) then
dbms_output.put_line
('Males and females are equal in number');
else
dbms_output.put_line
('females are less in count than males');
end if;
end proc_gender_cnt;
```

Ex: write a procedure to find the higher number of customers from the given 2 cities?

Ex: write a procedure to find the total balance amount in the given account type?

Ex: Write a procedure to display the number of products from the given 2 company names and find out from which company we have less number of products?

Ex: Write a procedure to display the number of mobile products from the given 2 company names?

Ex: Write a procedure to display the total number of customers opted for given account type?

Ex: write a procedure to display the higher total funds from the given 3 types of accounts?

LOOPS

Loops are used to execute the set of statements repeatedly.

Types: 3

1) simple loop

Syntax:

```
LOOP
stmt;
stmt;
:
EXIT  WHEN(condition);
stmt;
stmt;
:
END LOOP;
```

- No condition check before entering in to the loop
- Before exit from the loop it check condition, if the condition is true then loop execution terminates, if the condition fails then loop   execution is continue.

2) FOR LOOP

Syntax:

```
FOR <varname> IN [ REVERSE ] range_of_values
LOOP
stmt;
stmt;
:
END LOOP;
```

Ex:

```
for x in 1..5
loop
dbms_output.put_line(x);
end loop;
```

output:

```
1
2
3
4
5
```

Ex:

```
for x in REVERSE 1..5
loop
dbms_output.put_line(x);
end loop;
```

OUTPUT:

```
5
4
3
2
1
```

- No condition check before entering in to the loop
- The for loop variable need not be declared.
- In operator takes the values from first value to last value from the range
- Reverse operator enables IN to take the values from last to first

- Range is specified using ..(dot dot) operator.

3) WHILE LOOP

Syntax:

```
WHILE ( cond )
loop
stmt;
stmt;
:
end loop;
```

- Condition check before entering into the loop
- If the condition failed then the loop execution terminates.

Ex: write a procedure to display the sequence of values from 1 to 10 using above loops?

```
create or replace procedure proc_seqnos is

x        int:=1;
begin
--x:=1;
dbms_output.put_line(' By using  simple loop');
dbms_output.put_line('----------------------');
loop
dbms_output.put_line(x);
x:=x+1;
exit when(x > 10);
end loop;
DBMS_OUTPUT.PUT_LINE(' END  OF SIMPLE LOOP');
DBMS_OUTPUT.PUT_LINE('  ');

dbms_output.put_line(' By using for loop');
dbms_output.put_line('----------------------');

for y in 1..10
loop
dbms_output.put_line(y);
end loop;
DBMS_OUTPUT.PUT_LINE(' END  OF FOR LOOP');
DBMS_OUTPUT.PUT_LINE('  ');
```

--Display reverse numbers from 10 to 1 using For Loop

```
          dbms_output.put_line(' By using for loop , Displaying 1 to 10 values in reverse');
          dbms_output.put_line('----------------------------------------------------------');

          for y in REVERSE 1..10
          loop
          dbms_output.put_line(y);
          end loop;
          DBMS_OUTPUT.PUT_LINE(' END  OF FOR LOOP');
          DBMS_OUTPUT.PUT_LINE('  ');

dbms_output.put_line( ' current value of x: ' || x);


          dbms_output.put_line(' By using while loop');
          dbms_output.put_line('----------------------');
          x:=1;
dbms_output.put_line( ' New value of x: ' || x);
          while(x<=10)
          loop
          dbms_output.put_line(x);
          x:=x+1;
          end loop;
          DBMS_OUTPUT.PUT_LINE(' END  OF WHILE LOOP');
          end proc_seqnos;
```

Ex: write a procedure to display sequential even numbers from given number TO GIVEN NUMBER?

```
create or replace procedure proc_even(S INT,e  int)
is
r int;
nd int;
ST INT;
begin
ST:=S;
nd:=e;
if(st <= nd) then
dbms_output.put_line(' EVEN NUMBERS FROM  '||St||' TO '||nd);
dbms_output.put_line('**********************************');
while (st<=nd)
loop
select mod(st,2) into r from dual;
if(r=0) then
dbms_output.put_line('Even-'||st);
end if;
st:=st+1;
end loop;
else
dbms_output.put_line(' Invalid Range ');
end if;
end proc_even;
```

Ex:	write a procedure to insert the records like radius, pi value and area values	into the table	"circle"
for the radius range 1 to 10?

```
create table circle
(
radius  number(2),
pi      number(3,2),
area    number(6,2)
);

create or replace procedure proc_circle_area is
r       int;
pi      constant number(3,2):=3.14;
a       number(7,2);
begin
r:=1;
for x in 1..10
loop
a:=pi*r*r;
insert into circle values(r,pi,a);
r:=r+1;
end loop;
end proc_circle_area;
```

sample output:

```
exec proc_circle_area;

select * from circle;




anonymous block completed
   RADIUS         PI       AREA
---------- ---------- ----------
        1       3.14       3.14
        2       3.14      12.56
        3       3.14      28.26
        4       3.14      50.24
        5       3.14       78.5
        6       3.14     113.04
        7       3.14     153.86
        8       3.14     200.96
        9       3.14     254.34
       10       3.14        314

 10 rows selected
```

Ex: write a procedure to display reverse number for the given number?

```
create or replace procedure proc_revno(n  number)
is
rn  number;
begin
for x in reverse 1..length(n)
loop
rn:=rn||substr(n,x,1);
end loop;
dbms_output.put_line(' Given number= '||n);
dbms_output.put_line(' Reverse number= '||rn);
end proc_revno;
```

n	1..length(n)	x	substr(n,x,1)	rn||substr(n,x,1)	rn
6329	1..4	4	substr(6329,4,1)	||9	9
6329	1..3	3	substr(6329,3,1)	9||2	92
6329	1..2	2	substr(6329,2,1)	92||3	923
6329	1..1	1	substr(6329,1,1)	923||6	
	9236				

EX:

```
--display employee names who is getting the given salary?

create or replace procedure proc_ename(vsal  number)
is
vename  varchar2(10);
begin
select ename into vename from emp where sal=vsal;
dbms_OUTPUT.PUT_LINE(VENAME);
END PROC_ENAME;

EXEC PROC_ENAME(10000);
```

ERRORS LIKE:

```
01422. 00000 -  "exact fetch returns more than requested number of rows"

01403. 00000 -  "no data found"

....
```

NOTE:
General programs or procedures cannot be able to display multiple records from a table or multiple values from a column.

For Multiple records-----Explicit cursors

For multiple values from a column---Using VARRAYS

EXCEPTIONS

Exception is known as an error which can be handled by the user by displaying user friendly error messages.

Exception types: 2

i) Pre defined exceptions / Named Exceptions

These can be managed by data base engine as follows.

- While execution if there exists any error then it will look for exception block
- With in that it will look for corrsponding exception handler
- Then it executes the stmts with in the exception handler.

Exception handlers:

i) TOO_MANY_ROWS
It can be raised when the select stmts is selecting data from multiple records.

ii) NO_DATA_FOUND
It can be raised if there is no matched data from the table.

iii) ZERO_DIVIDE
It can be raised if a value is devided by zero

iv) CURSOR_ALREADY_OPEN
It can be raised if we are trying to open a cursor which is already opened.

v) VALUE_ERROR
It can be raised if there is incompatibility between declared variable and recieving value by that variable.

EXAMPLE PROCEDURES:

Ex:
Write a procedure to display employee names and job for the given salary?

```
CREATE OR REPLACE PROCEDURE PROC_no_EXCEP(VSAL  NUMBER)
```

```
IS
VNAME EMP.ENAME%TYPE;
VJOB  EMP.JOB%TYPE;
BEGIN
SELECT ename,job INTO vname,vjob from emp where sal=vsal;
dbms_output.put_line('Name of emp:'||vname||';  Desg of emp:  '||vjob);
dbms_output.put_line(' End of procedure ');
end proc_no_excep;

SET SERVEROUTPUT ON;

EXEC PROC_NO_EXCEP(5000);

SELECT ENAME ,JOB ,SAL FROM EMP;

EXEC PROC_NO_EXCEP(800);

EXEC PROC_NO_EXCEP(10000);
```

Ex: Above procedure with exceptions

```
CREATE OR REPLACE PROCEDURE PROC_EXCEP(VSAL  NUMBER)
IS

VNAME EMP.ENAME%TYPE;
VJOB  EMP.JOB%TYPE;

BEGIN

SELECT ename,job INTO vname,vjob from emp where sal=vsal;
dbms_output.put_line('Name of emp:  '||vname||'   ;   Job of emp:   '||vjob);
dbms_output.put_line(' End of procedure ');

EXCEPTION
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT.PUT_LINE(' SAL IS DUPLICATED');
DBMS_OUTPUT.PUT_LINE(' END OF TOO_MANY_ROWS--Exception');
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('REQUIRED SAL IS NOT AVAILABLE');
DBMS_OUTPUT.PUT_LINE(' END OF NO DATA FOUND');

end proc_excep;


EXEC PROC_EXCEP(800);----Only one employee is getting sal-display output

EXEC PROC_EXCEP(8000);---No emp getting this salary , so raise an error no_data_found
EXEC PROC_EXCEP(1250);---Multiple emps getting the salary, so raise an error
Too_many_rows.
```

Ex:
Write a procedure to display the employee details who is working with given designition?

```
      create or replace procedure proc_emp_dtls_desg(vjob        emp.job%type)
      is
      r_emp  emp%rowtype;
      begin
      select * into r_emp from emp where job=vjob;
      dbms_output.put_line(' The given job is '||vjob);
      dbms_output.put_line(r_emp.empno||' '||r_emp.ename||' '||r_emp.job||' '||r_emp.sal||
                                  '  '||r_emp.hiredate||' '||r_emp.comm||' '||r_emp.deptno);
    EXCEPTION
    WHEN TOO_MANY_ROWS THEN
     DBMS_OUTPUT.PUT_LINE(' MULTIPLE EMPS WORKING AS   '||VJOB);
     DBMS_OUTPUT.PUT_LINE(' I can display if there exists 1 emp with the given job,
                                                                Otherwise i am
      unable to display multiple records');
    WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('No EMPS WORKKING AS   '||VJOB);
      end proc_emp_dtls_desg;
```

sample outputs:

```
anonymous block completed
 MULTIPLE EMPS WORKING LIKE   SALESMAN

PROCEDURE PROC_EMP_DTLS_DESG compiled
anonymous block completed
 MULTIPLE EMPS WORKING LIKE   SALESMAN
 I can display if there exists 1 emp with the given job, Otherwise i am unable to display
multiple records

anonymous block completed
 The given job is PRESIDENT
7839 KING PRESIDENT 5000  17-NOV-81 2000 10
```

Ex: write a procedure to display customer account detials for the given customer id? If customer have multiple account or no account or no custid then display corresponding message?

---

2) User defined exceptions / Unnamed Exceptions

Here user has to manage the exceptions. User has to RAISE the exception if he is expecting any error. The exception is to raised by using exception variable name. The exception variable is acts as and EXCEPTION HANDLER in exception block.

i) declare exception type of variable in declare block

```
syn:    varname          exception;

Ex:     sal_miss     exception;
        comm_miss    exception;
        no_exp           exception;
```

ii) Raise the exception using the keyword RAISE in begin block

```
syn:    RAISE  < exptype varaiblename>;

Ex:     RAISE  COMM_MISS;
```

iii) Define the exception in exception block

```
syn:    WHEN  < exp handler> THEN
        stmt;
        stmts;
        :

ex:     when comm_miss then
        dbms_output.put_line(' comm value  is null');
```

Ex: Write a program to display commission of employee for the given employee id?

```
create or replace procedure proc_e_comm(veno  emp.empno%type) is
vcomm   int;
begin
select comm into vcomm from emp where empno=veno;
dbms_output.put_line('comm of '||veno||' is '||vcomm);
exception
when too_many_rows then
dbms_output.put_line(' Duplicate emp id existed ');
when no_data_found then
dbms_output.put_line(' Emp id not existed ');
end proc_e_comm;
```

Ex: write a procedure to display the commission of the employee for the given employee id?

```
    create or replace procedure proc_emp_comm( veno  emp.empno%type)
    is
    vcommemp.comm%type;
    comm_miss    exception; /* Declaring Exception type variable , Then is acting as
exception Handler.*/
    begin                 /* we can define exception handler in exception block */
    select comm into vcomm from emp where empno=veno;

    if vcomm is null then
    RAISE comm_miss;
```

```
            else
            dbms_output.put_line(' comm of  '||veno||'  is  '||vcomm);
            end if;
            exception
            when no_data_found then
            dbms_output.put_line(' No employee with  id'||veno);
            when Too_many_rows then
            dbms_output.put_line(' Duplicate empid existed');
            when comm_miss then
            dbms_output.put_line(' Employee  not getting commission ');
            end proc_emp_comm;
```

sample output:

```
PROCEDURE PROC_EMP_COMM compiled
anonymous block completed
 comm of  7654  is  1400

anonymous block completed
 comm of  7788  is  2000

anonymous block completed
 comm of employee is null

anonymous block completed
 No employee with given id
```

Ex: Write a procedure to display employee information , if any column value is null display corresponding message for the given employee id?

```
create or replace procedure proc_e1_comm1( veno  emp.empno%type)
            is
            r emp%rowtype;
  begin
            select * into r from emp where empno=veno;
  dbms_output.put_line(r.ename||';'||r.sal||';'||r.job||';'||r.hiredate);
  if r.mgr is null then
  dbms_output.PUt_line(' Mgr code is null for empid '||veno);
  else
  dbms_output.put_line(' mgr code is '||r.mgr);
  end if;
  if r.comm is null then
  dbms_output.PUt_line(' comm is null for empid '||veno);
  else
  dbms_output.put_line(' comm is '||r.comm);
  end if;
  if r.deptno is null then
  dbms_output.PUt_line(' deptno is null for empid '||veno);
  else
  dbms_output.put_line(' deptno is '||r.deptno);
  end if;
            exception
            when no_data_found then
```

```
          dbms_output.put_line(' No employee with  id'||veno);
          when Too_many_rows then
          dbms_output.put_line(' Duplicate empid existed');
                    end proc_e1_comm1;
```

----

ex: write a procedure to display the mfg, exp and warrenty of the given product id?
If any one is null then display corresponding message?

Ex: write a procedure to dispaly the account number,account type, account balance, account open date for given customer id?

CURSORS

QUERY EXECUTION PROCESS:

```
query------> sql stmt executor--> Oracle engine--> oracle DB--> DD-->actual data-->
          ----> temp memory area ( CURSOR )--->cache memory-----> CLIENT
                                                            [ from view ]
```

In the above the required data can be processed in the temporary memory area (cursor), and then displayed to the client . This is the case if we are selecting the data from the table.

After cursor memory , it maintains a copy of data in cache, if we are selecting the data from view.

CURSORS
It is a temp memory area, is used to perform intermediate operations before output displayed to the client.

Types of cursors: 2

1) Implicit cursors

A cursor which is used by sql query. This type can be assigned and managed by DB engine.

Properties:

SQL%ISOPEN	It returns true if the file is opened for processing
SQL%FOUND	It returns true if the matched data found
SQL%NOTFOUND	It returns true if there is no matched data
SQL%ROWCOUNT	It returns an integer value which represents the number of affected records.

2) Explicit cursors

These are declared and defined and managed by user. Generally we can use explicit cursors to display multiple records from the table through programs or procedures.

This process consists the following steps.

--declare and define the cursor in DECLARE block

```
SYN:    CURSOR  <cursor_name>  IS  SELECT ....... from  ...... where....order by......;

Ex:     cursor c_mgr   is select * from emp where job='MANAGER';
```

--Open the cursor for processing in BEGIN block

```
syn:    OPEN <cur_name>;

Ex:     OPEN c_mgr;
```

--Fetch data from the cursor in the begin block

```
syn:    FETCH <cur_name> INTO <varname>;
```

This fetch operation should be done repeatedly until all records are processed in, the cursor.

```
Ex:     LOOP
        FETCH c_mgr INTO REC;
        stmt;
        stmt;
        EXIT  WHEN (c_mgr%NOTFOUND);
        stmt;
        stmt;
        END LOOP;
```

--Close the cursor , to release the memory occupied by the cursor.

syn: CLOSE <cur_name>;

Ex: close c_mgr;

Explicit cursor properties

<cur_name>%ISOPEN	It returns true if the file is opened for processing
<cur_name>%FOUND	It returns true if there exits matched data
<cur_name>%NOTFOUND	It returns true if there is no matched data
<cur_name>%ROWCOUNT	It returns an integer value represents number of effected records.

Ex:
write a procedure to find the number of records affected if the emps are updated for the given job category with 10% of increment in their salary with in the table emp_copy?

```
create or replace procedure proc_update( vjob emp_copy.job%type)
is
begin
update emp_copy  set sal=sal+(0.10*sal) where job=vjob;
if (sql%found) then
dbms_output.put_line(' updation is success');
dbms_output.put_line(' number of records updated='||sql%rowcount);
else
dbms_output.put_line('updation not performed');
dbms_output.put_line(' number of records updated='||sql%rowcount);
end if;
end proc_update;
```

sample output:

Ex: write a procedure to display the employee details from given deptno?

```
/* Procedure with Explicit Cursor */

create or replace procedure proc_emp_dtls_dno (vdno       emp.deptno%type)
is
cursor c is select * from emp where deptno=vdno;
e_rec   emp%rowtype;
begin
open c;
dbms_output.put_line(' Details of emps under the deptno: '||vdno);
```

```
        dbms_output.put_line('------------------------------------------------------');
        loop
        fetch c into e_rec;
        exit when c%notfound;
        dbms_output.put_line(e_rec.empno||';'||e_rec.ename||';'||e_rec.sal||';'||e_rec.job||';'||e_
rec.deptno);
        end loop;
        close c;
        end proc_emp_dtls_dno;
```

CURSOR BASED RECORD VARIABLE

If we are fetching data from multiple tables multiple records, then we need to declare cursor based record variable . It supports all columns from the cursor.

syn: varname <cursor_name>%rowtype;

Ex: write a procedure to display the employee details along with his dept info from given deptno?

```
        /* Procedure with Explicit Cursor */

        create or replace procedure proc_emp_dept_dtls_dno (vdno
        emp.deptno%type)
        is
        cursor c is select e.empno,e.ename,e.sal,e.job,d.deptno,d.dname,d.loc from emp e,
dept d
                        where e.deptno=vdno and e.deptno=d.deptno;

        /* DECLARING CURSOR BASED RECORD type variable, Which SUPPORTS ALL
COLUMNS FROM THE CURSOR */

        /* IT IS REQUIRED IF WE ARE SELECTING DATA FROM MULTIPLE TABLES IN
TO THE CURSOR     */

        /* Declaring Cursor type variable-->        varname    cur_name%rowtype;        */

        c_rec   c%rowtype;
        begin
        open c;
        dbms_output.put_line(' Details of emps under the deptno: '||vdno);
        dbms_output.put_line(' -----------------------------------');
        loop
        fetch c into c_rec;
        exit when c%notfound;
        dbms_output.put_line(c_rec.empno||';'||c_rec.ename||';'||c_rec.sal||';'||c_rec.job||';'||c_r
ec.deptno||';'||c_rec.dname||';'||c_rec.loc);
        end loop;
        close c;
        end proc_emp_dept_dtls_dno;
```

Ex:	write a procedure to display the employee details of given job?

```
	create or replace procedure proc_emp_info_with_givenjob( vjob  emp.job%type)
	is
	cursor c_emp  is select * from emp where job=vjob;
	r	emp%rowtype;

	begin
	open c_emp;
	dbms_output.put_line(' the given job title:  '||vjob);
	dbms_output.put_line('all  '||vjob||' information');
	LOOP
	FETCH c_emp INTO r;
	exit when (c_emp%notfound);
	dbms_output.put_line(r.empno||' '||r.ename||' '||r.job||' '||r.sal||' '||r.hiredate||'
'||r.deptno);
	end loop;
	close c_emp;
	end proc_emp_info_with_givenjob;
```

sample output:

```
PROCEDURE PROC_EMP_INFO_GIVENJOB compiled
anonymous block completed
 the given job title:  MANAGER
all  MANAGER information
7566 JONES MANAGER 3272.5 02-APR-81 20
7698 BLAKE MANAGER 3135 01-MAY-81 30
7782 CLARK MANAGER 2695 09-JUN-81 10

PROCEDURE PROC_EMP_INFO_GIVENJOB compiled
anonymous block completed
 the given job title:  SALESMAN
all  SALESMAN information
7499 ALLEN SALESMAN 1600 20-FEB-81 30
7521 WARD SALESMAN 1250 22-FEB-81 30
7654 MARTIN SALESMAN 1250 28-SEP-81 30
7844 TURNER SALESMAN 1500 08-SEP-81 30
```

Ex:	write a procedure to display the employee names and salaries, their department names and locations?

```
	create or replace procedure proc_emp_dept is
	cursor c1 is select e.ename,e.sal,d.dname,d.loc from  emp e,dept d where
e.deptno=d.deptno;
	cur_var		c1%rowtype; /* DECLARING CURSOR TYPE VARIABLE */
	begin
	open c1;
	dbms_output.put_line('EMP-NAME		EMP-SALARY	Department name
	department-Location');
	dbms_output.put_line('=========		==========   ===============
	==================');
```

```
        LOOP
        FETCH c1 INTO cur_var;
        EXIT WHEN (c1%notfound);
        dbms_output.put_line(cur_var.ename||'                    '||cur_var.sal||'
        '||cur_var.dname||
                      '              '||cur_var.loc);
        end loop;
        close c1;
        end proc_emp_dept;
```

NOTE:
cursor type variable will supports all columns with in the cursor. No matter the columns are from which table.

sample output:

```
EMP-NAME          EMP-SALARY Department name          department-Location
=========         ========== ===============          ===================
SMITH          800               RESEARCH             DALLAS
ALLEN          1600              SALES           CHICAGO
WARD           1250              SALES           CHICAGO
JONES               2975              RESEARCH               DALLAS
MARTIN              1250              SALES           CHICAGO
BLAKE          2850              SALES           CHICAGO
CLARK               2450              ACCOUNTING             NEW YORK
SCOTT               3000              RESEARCH               DALLAS
KING           5000              ACCOUNTING           NEW YORK
TURNER              1500              SALES           CHICAGO
ADAMS               1100              RESEARCH               DALLAS
JAMES               950                   SALES       CHICAGO
FORD           3000              RESEARCH             DALLAS
MILLER              1300              ACCOUNTING             NEW YORK
dinesh         3400              OPERATIONS           BOSTON
```

```
--Write a procedure to display customer name,city, actno, balance,act_name of all
customers?

create or replace procedure proc_cust_act_info
is
cursor c_acts is select cd.cname,cd.city,cad.actno,cad.act_bal,at.act_name
            from cust_dtls cd, cust_act_dtls cad, act_types at
              where cd.cno=cad.cust_code  and cad.act_type=at.act_type;

rec c_acts%rowtype;  /* Declaring a cursor based variable , supports all columns from cursor
*/
begin
open c_acts;
dbms_output.put_line(' Customers and their Accounts information ');
dbms_output.put_line('CustName---custCity--- Account No:   Account Bal    Account Name');
dbms_output.put_line('---------------------------------------------------------------');
loop
```

```
fetch c_acts into rec;
EXIT when (c_acts%notfound);
dbms_output.put_line(rec.cname||'----'||rec.city||'---'||rec.actno||'    '||rec.Act_bal||'
'||rec.act_name);
end loop;
close c_acts;
end proc_cust_act_info;

exec proc_cust_act_info;
```

FUNCTIONS

It is database object which is used to perform a task and it should return a value to the calling program.

- Functions are stored as pre compiled object.
- Functions are also known as sub programs.
- Functions are calling from other procedure or functions or by using select stmts.
- Functions are may or may not take the arguments
- FUNCTIONS ARE IMPLICITLY EXECUTED.
- FUNCTIONS SHOULD return a value to the calling program.

Function also contains 2 parts. i.e Specification and Body like procedure.

Syntax

```
    create or replace function <func_name> ( argvar  datatype, argvar  datatype...... )
    RETURN <datatype>
    IS

    <declaration stmts>;
    begin
    <assignment stmts>;
    <data processing stmts>;
    <output stmts>;
    Exception
    <error handling stmts>;
    ......
    RETURN(value/var_name);
```

```
        END <func_name>;
```

Ex:

```
        create or replace function f_add( x  int,  y int)
        return   number
        is
        res  int;
        begin
        res:=x+y;
        return(res);
        end f_add;
```

Note:
A Function can be called through any SELECT or PROGRAM or Procedure.

Ex:

```
        select func_add(100,200) from dual;
        300
```

Ex:

CALLING FUNCTION BY USING PROCEDURE:-

```
 create or replace procedure p_add( a int, b int)
 is
   res int;
 begin
 res:=f_add(a,b);
 dbms_output.put_line(' addition='||res);
 end p_add;

 exec p_add(20,30);

 select f_add(100,400) from dual;



 declare
 s1 int;
 s2 int;
 r int;
 begin
 s1:=&s1;
 s2:=&s2;
  dbms_output.put_line(' calling function from a procedure through program');
   dbms_output.put_line(' ');
 p_add(s1,s2);  /* Procedure calling stmt */
  dbms_output.put_line(' completed');
   dbms_output.put_line('------');
    dbms_output.put_line(' calling function from program directly');
    r:=f_add(s1,s2); /* function calling stmt */
 dbms_output.put_line(' result='||r);
  dbms_output.put_line(' completed');
  end;
```

Example:

```
create or replace function f_add( x  int,  y int)
        return   number
        is
        res  int;
        begin
        res:=x+y;
        return(res);
        end f_add;

/* calling function using SELECT */

select f_add(32,23) from dual;


/* calling function using PROCEDURE */

create or replace procedure proc_fsample(x int, y int) is
r int;
begin
r:=f_add(x,y);
dbms_output.put_line(' output from function is '||r);
end proc_fsample;

exec proc_fsample(10,20);

/* calling function using PROGRAM OR ANONYMOUS BLOCK */

declare
a int;
b int;
output int;
begin
a:=&a;
b:=&b;
output:=f_add(a,b);
dbms_output.put_line(' result from function='||output);
end;
```

Ex: write procedure to display esal,bonus value and final salary (basic sal+ bonus) for the given employee id?

Ex: Do the above for all employees?

Ex: Do the above for all given job category of emps?

[Common Functionality is " Finding Bonus Based on salary Range"]

Bonus is to be calculated as follows

sal<1000	2%
>=1000 &< 2000	5%
>=2000 &<3000	10%
>=3000 &<5000	20%
>=5000	25%

*** Creating a function to calculate the Bonus?

```
create or  replace function func_bonus(s   emp.sal%type)
return number
is
bonus  number(7,2);
begin
if (s <1000) then
bonus:=0.02*s;
end if;
if (s>=1000 and s<2000) then
bonus:=0.05*s;
end if;
if (s>=2000 and s<3000) then
bonus:=0.10*s;
end if;
if (s>=3000 and s<5000) then
bonus:=0.20*s;
end if;
if (s>=5000) then
bonus:=0.25*s;
end if;
return(bonus);
end func_bonus;
```

```
create or replace procedure proc_emp_fsal(veno     emp.empno%type)
is
vsal     emp.sal%type;
b        number(7,2);
fsal     number(7,2);
begin
select sal into vsal from emp where empno=veno;
b:=func_bonus(vsal); /* function calling stmt */
fsal:=vsal+b;
dbms_output.put_line(' The salary details of  '||veno);
dbms_output.put_line('--------------------------------------------------');
dbms_output.put_line('  salary                    bonus            final salary');
dbms_output.put_line(  vsal||'        '||b||'                 '||fsal);
end proc_emp_fsal;
```

Ex:	Perform above for all emps:

```
create or replace procedure proc_emps_fsal
is
cursor c is select empno,sal from emp;
veno emp.empno%type;
vsal     emp.sal%type;
b        number(7,2);
fsal     number(7,2);
begin
open c;
dbms_output.put_line('veno    salary           bonus            final salary');
dbms_output.put_line('-----------------------------------------------------');
loop
fetch c into veno,vsal;
b:=func_bonus(vsal); /* function calling stmt */
fsal:=vsal+b;
exit when (c%notfound);
dbms_output.put_line(  veno ||' '||vsal||'        '||b||'               '||fsal);
end loop;
close c;
end proc_emps_fsal;
```

Ex: write a procedure to display the above output for all emps with in given job category?

```
create or replace procedure proc_emps_fsal_job(vjob  emp.job%type)
is
cursor c is select empno,sal from emp where job=vjob;
veno emp.empno%type;
vsal     emp.sal%type;
b        number(7,2);
fsal     number(7,2);
begin
open c;
dbms_output.put_line('veno    salary           bonus            final salary');
dbms_output.put_line('-----------------------------------------------------');
loop
fetch c into veno,vsal;
b:=func_bonus(vsal); /* function calling stmt */
fsal:=vsal+b;
exit when (c%notfound);
dbms_output.put_line(  veno ||' '||vsal||'        '||b||'               '||fsal);
end loop;
close c;
end proc_emps_fsal_job;
```

Ex: write a procedure and functions to display the productname, cost, discount, final price for the given product id?

Ex: write a procedure and functions to display the productname, cost, discount, final price for all products?

Ex: write a procedure and functions to display the productname, cost, discount, final price for the given category of products?

Discount calculation is as follows

cost< 5000	no discount
>=5000 &< 10000	5%
>=10000 &< 20000	10%
>=20000 &< 40000	20%
>40000	30%

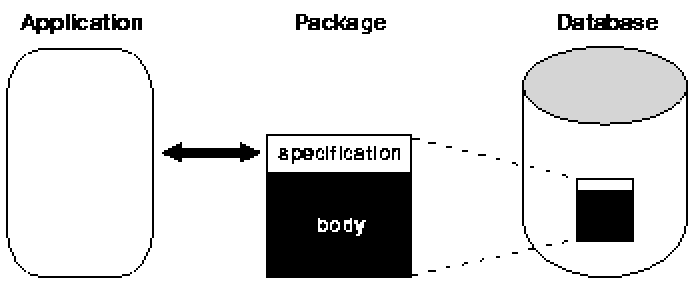
How do i delete function?

drop function <func_name>;

PACKAGES

It is a data base object which logically groups the related objects. It reduces the search time for the required object according to required action.

Package Interface



It contains 2 parts

1) package specification:

It contains function calling stmts and procedure calling stmts , global variable declarations.

syn-1: package specification

```
create or replace package <pkg_name> AS
<variable declarations>;
<procedure calling stmts>;
<function calling stmts>;
end <pkg_name>;
```

2) package body:
It contains functions and procedures functionalities or coding.

syn-2: package body

```
create or replace package body <pkg_name> IS
<procedure body>;
<function body>;
end <pkg_name>;
```

Ex: write a package which contains the procedures and functions to calculate sal, bonus, final salaries for given empid , for all emps and for given job category employees?

```
CREATE OR REPLACE PACKAGE PKG_Emp_FSAL
 AS
PROCEDURE PROC_EMP_FSAL(VENO  EMP.EMPNO%TYPE);
PROCEDURE PROC_EMPs_Fsal;
PROCEDURE PROC_EMPs_FSAL_JOB( vjob  emp.job%type);
FUNCTION  FUNC_bonus(S EMP.SAL%TYPE)
RETURN  NUMBER;
END PKG_EMP_FSAL;

/

package created.
```

EX:package body

```
CREATE OR REPLACE PACKAGE BODY PKG_EMP_FSAL IS

/* PROC_EMP_FSAL  */
```

```
        Procedure proc_emp_fsal(veno       emp.empno%type)
        is
        vsal     emp.sal%type;
        b        number(7,2);
        fsal     number(7,2);
        begin
        select sal into vsal from emp where empno=veno;
        b:=func_bonus(vsal); /* function calling stmt */
        fsal:=vsal+b;
        dbms_output.put_line(' The salary details of  '||veno);
        dbms_output.put_line('--------------------------------------------------');
        dbms_output.put_line
('  salary                 bonus             final salary');
        dbms_output.put_line
(  vsal||'        '||b||'                '||fsal);
        end proc_emp_fsal;


        /* function to calcualte the BONUS */

        Function func_bonus(s   emp.sal%type)
        return number
        is
        bonus  number(7,2);
        begin
        if (s <1000) then
        bonus:=0.02*s;
        end if;
        if (s>=1000 and s<2000) then
        bonus:=0.05*s;
        end if;
        if (s>=2000 and s<3000) then
        bonus:=0.10*s;
        end if;
        if (s>=3000 and s<5000) then
        bonus:=0.20*s;
        end if;
        if (s>=5000) then
        bonus:=0.25*s;
        end if;
        return(bonus);
        end func_bonus;

        /* Procedure to display the sal info of all emps */

        Procedure proc_emps_fsal
        is
        cursor c is select empno,sal from emp;
        veno    emp.empno%type;
        vsal     emp.sal%type;
        b        number(7,2);
        fsal     number(7,2);
        begin
        open c;
```

```
          dbms_output.put_line('---------------------------------------------------');
          dbms_output.put_line
('  salary                 bonus            final salary');
          dbms_output.put_line('**********************************************');
          loop
          fetch c into veno,vsal;
          dbms_output.put_line(' The salary details of  '||veno);
          b:=func_bonus(vsal); /* function calling stmt */
          fsal:=vsal+b;
          EXIT WHEN (C%NOTFOUND);
          dbms_output.put_line
(  vsal||'        '||b||'                 '||fsal);
          END LOOP;
          CLOSE C;
          end proc_emps_fsal;


                    /* procedure  proc_emps_fsal_job  */

          procedure proc_emps_fsal_job(vjob emp.job%type)  is
          cursor c is select empno,sal from emp where job=vjob;
          vsal     emp.sal%type;
          veno     emp.empno%type;
          incrval  number(7,2);
          fsal     number(7,2);
          begin
          open c;
          loop
          fetch c into veno,vsal;
          incrval:=func_bonus(vsal);  /* function calling stmt  */
          fsal:=vsal+incrval;


          EXIT WHEN c%NOTFOUND;
          dbms_output.put_line(' sal,incrval,final salary of  '||veno);
          dbms_output.put_line(vsal||' , '||incrval||' , '||fsal);
          end loop;
          close c;
          end proc_emps_fsal_job;

          END PKG_EMP_FSAL;

          /
```

HOW TO EXECUTE A PROCEDURE FROM A PACKAGE?

syn: `exec package_name.proc_name(argval, argval......);`

Ex: `exec pkg_emp_fsal.proc_emp_fsal(7654);`

How to drop a package?

```
drop package  <pkg_name>;
```

Ex: write a procedure to display the final product cost
  a) for the given product-id,
  b) for all products
  c) given category of products ?

The discount is applied based on following ranges?

<5000	5%
>=5000 &<10000	10%
>=10000 &<20000	15%
>=20000	20%

Ex: create a package for the above?

Ex: write a procedure to display the final amount paid to the customer by adding interest amount for the principal amount?
Do the above process for given customer id?
Do it for all customers?
Do it for given type of account holders?

The interest is calculated based on the following?

<1000	no interest
>=1000 &< 25000	7.5%
>=25000 &< 50000	8.00%
>=50000 &< 100000	9.00%
>=100000 &<=600000	10.00%
>600000	Not applicable

COLLECTIONS

Collection means a group of values.
If a variable is able to store group of values at a time then it is known as Collection Type variable.

Ex: Table based record variable.
User defined record variable.
Array type variable.

TYPES OF ARRAYS:

VARRAYS [Variable sized Array]
ASSOCIATIVE ARRAYS [No size specification]
NESTED TABLES [Multidimensional Array]

VARRAYS--Single Dimensional Array

Variable sized array. Array is able to store multiple values with same data type at continuous memory locations. Here we must specify the size. and the specified size is fixed.

<u>Defining array strucutre:</u>

```
TYPE  <typename>  IS VARRAY(size) OF <Array_element_Data_type>;

Ex:     type  empsal  is varray(20) of number;

                        or

        type  empsal  is varray(20) of emp.sal%type;
```

How do i declare array type variable?

```
        VARNAME             Array_type_name;

Ex:     e1      empsal;
        e2      empsal;
```

For the Array variable e1 , the memory assigned is as follows.
Each memory location is identified with SUBSCRIPTS value. Generally subscripts are integers begining with 1,....

```
                                    e1
                                    |
        ------------------------------------------------------------------------------
        |       |       |       |       |       |       |       |       |.......20 locations

subscripts   1      2       3       4  { Array location Identifiers}
```

How do i assign values into array type variable?

```
syn:            array_var_name          := array_name(val1,val2,.......);

Ex:             e1:=empsal(2300,3400,2000,1200,......);

                                    e1
```

```
                                        |
    ----------------------------------------------------------------------------------
    |2300   |3400   |2000   |1200   |         |         |         |         |.......20 locations

      1          2        3        4  { Array location Identifiers}
```

Accessing / displaying values from Array type variable:

syn: Array_var_name(subscriptNumber)

Ex:

e1(3)
e1(1)

By using any loop:

```
declare
x         int:=1;

for i in e1.first..e1.last
loop
dbms_output.put_line(e1(x));
x:=x+1;
end loop;
```

Methods related with arrays:

COUNT--Holds number of array locations
FIRST--takes value from first location
LAST--indicates last location of array
NEXT--indicates next location
PRIOR--Indicates previous location

Ex: Display employee names and salaries using VARRAY?

```
declare
type  esal  is varray(5) of number;
type  enames is varray(5) of varchar(10);
vsal     esal;
vename          enames;
x         int:=1;
begin

vsal:=esal(1250,4500,2300,7600,1000);
vename:=enames('Ajay','Bhagath','Xavier','SMITH','SCOTT');

/* displaying data from array type variable */
```

```
for a in VSAL.FIRST..VSAL.LAST  --[ 1..VSAL.COUNT ]
loop
dbms_output.put_line(vename(x)||' is getting sal: '||Vsal(x));
x:=x+1;
end loop;
dbms_output.put_line(' end of program');
end;
```

Note:

In the above examples,

esal.FIRST .. esal.LAST --Represents from the first location of vsal to the last location of vsal.

The following are methods used along with Array type variables.

FIRST
LAST
NEXT
PRIOR
COUNT

BULK COLLECT

It improves the performance of database. It eliminates the context switching between program area and cursor area.

By using this we can take all values from the cursor at a time and control coming to program area and save that collected values in to array type variable.

Ex:
Write a procedure to Display all employee names with out BULK COLLECT and varrays?

```
create or replace procedure proc_enames
is
cursor c is select ename from emp;
vename emp.ename%type;
begin
open c;
dbms_output.put_line(' All Employee Names ');
dbms_output.put_line('====================');
loop
fetch c into vename;
exit when c%notfound;
dbms_output.put_line(vename);
end loop;
close c;
end proc_enames;
```

Ex: write a procedure to display employee names from emp using varrays and bulk collect?

BULK COLLECT:

It reduces number of context switches from program Memory to cursor Memory and vice versa. There by it is Increasing DB performance.

Ex: write a procedure to display employee names with BULK COLLECT

```
create or replace procedure proc_names_sal is

/* using explicit cursors */

        cursor c is select ename,sal from emp;

/* creating array type structure for storing employee names */


        type enames is varray(25) of emp.ename%type;

/* creating array type structure for storing employee salaries */

        type esal is varray(25) of emp.sal%type;

        vename          enames;
        vsal    esal;
        x         int:=1;
        begin

        open c;

        fetch c  bulk collect     into vename,vsal;
        close c;

/* displaying data from array type variable */

        dbms_output.put_line(' All Employee Names and salaries ');
        dbms_output.put_line('====================================');

        for i in Vename.FIRST..Vename.LAST  --[ 1..VSAL.COUNT ]
        loop
        dbms_output.put_line
(vename(x)||' is getting salary is '||vsal(x));
        x:=x+1;
        end loop;
        dbms_output.put_line(' end of program');
        end proc_names_sal;


exec proc_names_sal;
```

sample output:

```
PROCEDURE PROC_NAMES compiled
anonymous block completed
SMITH
ALLEN
WARD
JONES
MARTIN...
```

ii) ASSOCIATIVE ARRAYS

It is also a single Dimensional array.
These arrays are not having any limit for size like VARRAYS.
we can also index the locations of Associative array with STRING VALUES. (Oracle 9i Release 2 onwards)

```
                                        Associative
                                             |
                    ---------------------------------------------------------
                    |        |        |        |        ........         N is size( NO specification of
size)

INT      subscripts  1       2          3.......................

String  subscripts   abc     xyz    stu
```

Ex:
    asso(1)
    asso(abc)

    Both represents first location of associative array.

```
syn:    TYPE  <type names >  IS TABLE OF <element type>
        INDEX BY PLS_INTEGER / BINARY_INTEGER /VARCHAR2;
```

Ex: Write a procedure to display employee names and salaries using Associative arrays, bulk collect?

```
create or replace procedure proc_name_sal is

/* using explicit cursors */

        cursor c is select ename,sal from emp;

/* creating array type structures for storing employee names and salaries */

        type  esal  is TABLE of emp.sal%type INDEX BY PLS_INTEGER;
        type  enames is TABLE of emp.ename%type INDEX BY PLS_INTEGER;
        vsal     esal;
        vename          enames;
        x         int:=1;
        begin

        open c;

        fetch c  bulk collect     into vename,vsal;
      close c;

      /* displaying data from array type variable */

        for i in Vename.FIRST..Vename.LAST  --[ 1..VSAL.COUNT ]
        loop
        dbms_output.put_line(vename(x)||' is getting sal :'||vsal(x));
        x:=x+1;
        end loop;
        dbms_output.put_line(' end of program');

        end proc_name_sal;

  exec proc_name_sal;
```

sample output:

```
anonymous block completed
SMITH is getting sal :800
ALLEN is getting sal :1600
WARD is getting sal :1250
JONES is getting sal :2975
MARTIN is getting sal :1250
BLAKE is getting sal :2850
CLARK is getting sal :2450
SCOTT is getting sal :3000
KING is getting sal :5000
TURNER is getting sal :1500
ADAMS is getting sal :1100
JAMES is getting sal :950
FORD is getting sal :3000
MILLER is getting sal :1300
 end of program
```

Ex:

Write a procedure to display all product names,cost using bulk collect and associative Arrays?

iii) Nested Tables: (multi dimensional array)

Used to store table like records in the array. Each array location devided in to parts. IN these parts we can store multiple data types of values.

To perform above, do the following.

a) creating an array of record type structure.

--define record type structure
--define array with record type structure

----> To define a PL/SQL type for nested tables, use the

Nested table syntax:

TYPE type_name IS TABLE OF rec_type_variablename [NOT NULL];

Ex: Display empid,ename and salaries of all emps with given job?

```
create or replace procedure proc_empjob_rec(vjob  emp.job%type) is

cursor c_emp is select empno,ename,sal from emp where job=vjob;

 TYPE e_rec IS RECORD(
                        eid     emp.empno%type,
                        name    emp.ename%type,
                        salary  emp.sal%type
                        );

TYPE e_arr IS TABLE OF e_rec INDEX BY BINARY_INTEGER;

emp_rec  e_arr;

x int:=1;

begin

open c_emp;
fetch c_emp bulk collect into emp_rec;
close c_emp;
dbms_output.put_line( 'Information of  '||vjob||'s');
dbms_output.put_line('----------------------------------------');

for i in 1..emp_rec.count
```

```
loop
dbms_output.put_line(emp_rec(x).eid||';'||emp_rec(x).name||';'||emp_rec(x).salary);
x:=x+1;
end loop;
end proc_empjob_rec;
```

EX:write a procedure to display pid,pname,cost,mfg,comp_name,comp_country for all products?

```
SET SERVEROUTPUT ON
DECLARE
  TYPE country_rec IS RECORD (
                                iso_code  VARCHAR2(5),
                                name      VARCHAR2(50)
                                );

/* Indexing by integer and no limit for the array  */

  TYPE country_arr IS TABLE OF country_rec
    INDEX BY BINARY_INTEGER;

                                        country_arr

              1                              2|              3
        ---------------------------------------------------------------------------------------
        |                    |                    |                    |.....
                 |                    |                    |

        iso_code       name    iso_code   name       iso_code    name


  v_country country_arr;
BEGIN


  v_country(1).iso_code := 'UK';
  v_country(1).name     := 'United Kingdom';
  v_country(2).iso_code := 'US';
  v_country(2).name     := 'United States of America';
  v_country(3).iso_code := 'FR';
  v_country(3).name     := 'France';
  v_country(4).iso_code := 'DE';
  v_country(4).name     := 'Germany';


  FOR i IN 1 .. 4
 LOOP
    IF v_country(i).iso_code = 'DE' THEN
      DBMS_OUTPUT.PUT_LINE('ISO code "DE" = ' || v_country(i).name);
    END IF;
  END LOOP;

END;
/
```

Ex:

```
/* Indexing Associative array by using string values  */

SET SERVEROUTPUT ON
DECLARE
  TYPE country_tab IS TABLE OF VARCHAR2(50)
    INDEX BY VARCHAR2(5);

  t_country country_tab;
BEGIN


  t_country('abc') := 'United Kingdom';
  t_country('xyz') := 'United States of America';
  t_country('FR') := 'France';
  t_country('DE') := 'Germany';

  -- Find country name for ISO code "DE"
  DBMS_OUTPUT.PUT_LINE('ISO code "DE" = ' || t_country('DE'));

END;




SET SERVEROUTPUT ON
DECLARE
  TYPE country_tab IS TABLE OF VARCHAR2(50)
    INDEX BY VARCHAR2(5);

  t_country country_tab;
BEGIN


  t_country('abc') := 'United Kingdom';
  t_country('xyz') := 'United States of America';
  t_country('zzz') := 'France';
  t_country('xxx') := 'Germany';

  -- Find country name for ISO code "DE"
  DBMS_OUTPUT.PUT_LINE('ISO code "zzz" = ' || t_country('zzz'));

END;
```

PL/SQL Procedure Parameters

IN, OUT and **IN OUT** parameters

```
create or replace procedure <name> ( Formal Parameters List )
is
.....
......
end <name>;
```

```
Exec <Proc_name>( Actual parameters....List);
```

<u>**IN** Parameter</u>

How to use IN parameter properly?
Here are the rules about IN parameters:

- A formal IN parameter acts like constant. It can not be assigned with new values.
- An actual IN parameter can take a value or a variable.
- An actual IN parameter is passed by reference to the specified value or the value of the specified variable.
- An actual IN parameter will not receive any value from the formal parameter.

Ex: write a procedure to display name of given employee id?

```
create or replace procedure proc_ename(veid  IN  emp.empno%type)
is
vename  varchar2(20);
begin
select ename into vename from emp where empno=veid;
dbms_output.put_line(' name of emp='||vename);
--veid:=7654;
end proc_ename;
```

Ex:

```
 CREATE OR REPLACE PROCEDURE WELCOME AS
   SITE CHAR(80) := 'FYICenter.com';
   PROCEDURE WELCOME_PRINT(S IN CHAR) AS
   BEGIN
     DBMS_OUTPUT.PUT_LINE('Welcome to ' || S);
     -- S := 'Google.com'; -- Not allowed
   END;
 BEGIN
   WELCOME_PRINT('MySpace.com');
   WELCOME_PRINT(SITE);
  END;
  /

SQL> EXECUTE WELCOME;
Welcome to MySpace.com
Welcome to FYICenter.com
```

<u>OUT</u> <u>Parameter</u>

How to use OUT parameter properly?

Here are the rules about OUT parameters:
- A formal OUT parameter acts like an un-initialized variable. It must be assigned with new values before the end of the procedure or function.
- An actual OUT parameter must be a variable.
- An actual OUT parameter will not pass any value to the formal parameter.
- An actual OUT parameter will receive a copy of the value from the formal parameter at the end of the procedure or function.

Ex:

```
 CREATE OR REPLACE PROCEDURE WELCOME AS
    SITE CHAR(40) := 'FYICenter.com';
    MESSAGE CHAR(80);
    PROCEDURE WELCOME_PRINT(S IN CHAR, M OUT CHAR) AS
    BEGIN
      M := 'Welcome to ' || S;
    END;
  BEGIN
    WELCOME_PRINT('MySpace.com', MESSAGE);
    DBMS_OUTPUT.PUT_LINE(MESSAGE);
    WELCOME_PRINT(SITE, MESSAGE);
   DBMS_OUTPUT.PUT_LINE(MESSAGE);
  END;
  /

SQL> EXECUTE WELCOME;
Welcome to MySpace.com
Welcome to FYICenter.com
```

Ex:

```
create or replace procedure proc_e_sal
(veid in emp.empno%type,vsal out emp.sal%type)
is
begin
select sal into vsal from emp where empno=veid;
end proc_e_sal;


/* calling Program */

declare
veid  emp.empno%type;
vsal  emp.sal%type;
begin
veid:=&veid;
proc_e_sal(veid,vsal);
```

```
dbms_output.put_line(' emp salary= '||vsal);
end;
```

Ex: write a procedure to "return" Name of employee for the given employee id?

```
        create or replace procedure proc_ename_out(veno IN int,vename OUT
emp.ename%type)
        is

        begin
        select ename into vename from emp where empno=veno;
        dbms_output.put_line(' Vename value returned to Calling Program');
        end proc_ename_out;
```

Ex: /* calling above Procedure from a Program */

```
        declare
        veid    int;
        empname      varchar2(20);
        Begin
        veid:=&veid;
        --Procedure calling stmt
        Proc_ename_out(veid,empname);
        dbms_output.put_line('name of '||veid|| ' is '||empname);
        end;
```

IN OUT Parameter

How to use IN OUT parameter properly?

Here are the rules about IN OUT parameters:
A formal IN OUT parameter acts like an initialized variable.
An actual IN OUT parameter must be a variable.
An actual IN OUT parameter passes a copy of its value to the formal parameter when entering the procedure or function.
An actual IN OUT parameter will receive a copy of the value from the formal parameter at the end of the procedure or function.

Ex:

```
 CREATE OR REPLACE PROCEDURE SWAP_TEST AS
   A NUMBER := 3;
   B NUMBER := 8;
  PROCEDURE MY_SWAP(X IN OUT NUMBER,Y IN OUT NUMBER) AS
     T NUMBER;
   BEGIN
     T := X;
     X := Y;
     Y := T;
   END MY_SWAP;
  BEGIN
   MY_SWAP(A,B);
   DBMS_OUTPUT.PUT_LINE('A = ' || TO_CHAR(A));
   DBMS_OUTPUT.PUT_LINE('B = ' || TO_CHAR(B));
  END;
  /

 SQL> EXECUTE SWAP_TEST;
 A = 8
 B = 3
```

TRIGGERS

TRIGGER is a data base object which can be enabled or executed implicitly for any dml operation on the table.

Generally it can be used to take a audit information of table data.

Trigger contains 3 parts:

i) trigger event
Any dml (insert or update or delete) operation is known as trigger event

ii) trigger restriction
It controls the trigger execution that is before or after dml operation.

iii)Trigger action
It implements or represent the logic or functionality of the trigger, that is what it is doing for dml operation.

Types of triggers: 12

At row level

FOR EACH ROW--It enable the trigger execution for each record inside the table.
The following are record level triggers

before insert
after insert
before update
after update
before delete
after delete

At statement level--Trigger is executed only once for any number affected records for a dml operation.

```
before insert
after insert
before update
after update
before delete
after delete
```

:NEW--It represents the new values into a column[insert & After update]
:OLD--It represents the old values from a column [Before update & before delete]

Syntax:

```
create or replace trigger <trig_name>
[before / after]
[ insert / update /delete ] of  <col1>,<col2>......
on  <table name>
FOR EACH ROW
declare
------
------
BEGIN
------------
------------
------------
END <trig_name>;

/

trigger created.
```

Ex: table

```
create table employee
(
ename  varchar2(20),
city  varchar2(20)
);

insert into employee values('Kiran','Delhi');
insert into employee values('madhu','Delhi');
insert into employee values('DINESH','HYD');
insert into employee values('smith','texas');

select * from employee;

ENAME                CITY
-------------------- --------------------
```

```
Kiran              Delhi
madhu                Delhi
DINESH               HYD
smith              texas
```

Ex: write a trigger to insert or update the data of employee in upper case?

[Generally to maintain uniform data in the business Database]

```
create or replace trigger trig_upper
before
insert or update  on employee
for each row
begin
:new.ename:=upper(:new.ename);
:new.city:=upper(:new.city);
end trig_UPPER;

/
```

Ex: write a trigger to maintain Audit transaction details of employee table data?

```
create table emp_audit_New
                                        (
                                        ename varchar2(20),
                                        city      varchar2(20),
                                        opname           varchar2(30),
                                        opdt     date
                                        );



create table emp_audit_Changed
          (
          ename varchar2(20),
          city      varchar2(20),
          opname           varchar2(30),
          opdt     date
          );

create table emp_audit_delete
          (
          ename varchar2(20),
          city      varchar2(20),
          opname           varchar2(30),
          opdt     date
          );


          create or replace trigger trig_emp_Audit
          before
          insert or update or delete  on employee
```

```
for each row
declare
opname          varchar2(32);

begin


if  inserting  then
opname:='inserted';

insert into emp_audit_New values(:new.ename,:new.city,opname,sysdate);
end if;

if updating then
opname:='Before Update-oldvalues';
insert into emp_audit_Changed values
(:old.ename,:old.city,opname,sysdate);

opname:='After Update--new values';
insert into emp_audit_Changed values
(:new.ename,:new.city,opname,sysdate);
end if;

if deleting  then
opname:='deleting';
insert into emp_audit_delete  values
(:old.ename,:old.city,opname,sysdate);
end if;

end trig_emp_Audit;

/
```

sample examples:

```
SQL> select * from employee;

ENAME      CITY
---------- ----------
A          CHENNAI
b          chennai
s          blore
X          DALLAS
Y          TEXAS

SQL> select * from trig_emp2;

ENAM CITY       OPNAME     OPDT
---- ---------- ---------- ---------
y    texas      insert     08-JAN-13

SQL> delete from employee where ename='b';

1 row deleted.
```

```
SQL> select * from trig_emp2;

ENAM CITY       OPNAME     OPDT
---- ---------- ---------- ---------
y    texas      insert     08-JAN-13
b    chennai    delete     08-JAN-13

SQL> select * from employee;

ENAME      CITY
---------- ----------
A          CHENNAI
s          blore
X          DALLAS
Y          TEXAS
```

NOTE:
system table: USER_TRIGGERS

It contains the information of triggers created in the database.

CASE statement:

The CASE statement begins with the keyword CASE. The keyword is followed by a selector. The selector expression can be arbitrarily complex. For example, it can contain function calls. Usually, however, it consists of a single variable. The selector expression is evaluated only once. The value it yields can have any PL/SQL datatype other than BLOB, BFILE, an object type, a PL/SQL record, an index-by-table, a varray, or a nested table.

The selector is followed by one or more WHEN clauses, which are checked sequentially. The value of the selector determines which clause is executed. If the value of the selector equals the value of a WHEN-clause expression, that WHEN clause is executed.

The ELSE clause works similarly to the ELSE clause in an IF statement. In the last example, if the grade is not one of the choices covered by a WHEN clause, the ELSE clause is selected, and the phrase 'No such grade' is output. The ELSE clause is optional. However, if you omit the ELSE clause, PL/SQL adds the following implicit ELSE clause:

```
ELSE RAISE CASE_NOT_FOUND;
```

If the CASE statement selects the implicit ELSE clause, PL/SQL raises the predefined exception CASE_NOT_FOUND. So, there is always a default action, even when you omit the ELSE clause.

The keywords END CASE terminate the CASE statement. These two keywords must be separated by a space. The CASE statement has the following form:

SYNTAX:

```
CASE selector
  WHEN expression1 THEN sequence_of_statements1;
  WHEN expression2 THEN sequence_of_statements2;
  ...
  WHEN expressionN THEN sequence_of_statementsN;
 [ELSE sequence_of_statementsN+1;]
END CASE;
```

Example:

```
        declare
        grade   char;
        begin
        dbms_output.put_line(' Select anyone Grade from the following');
        dbms_output.put_line(' A B C D F');
        grade:='&grade';

CASE grade
  WHEN 'A' THEN dbms_output.put_line('Excellent');
  WHEN 'B' THEN dbms_output.put_line('Very Good');
  WHEN 'C' THEN dbms_output.put_line('Good');
  WHEN 'D' THEN dbms_output.put_line('Fair');
  WHEN 'F' THEN dbms_output.put_line('Poor');
  ELSE dbms_output.put_line('Invalid grade selection');
END CASE;
END;
```

An alternative to the CASEstatement is the CASE expression, where each WHEN clause is an expression.

Searched CASE Statement:-

PL/SQL also provides a searched CASE statement, which has the form:

```
CASE
  WHEN search_condition1 THEN sequence_of_statements1;
  WHEN search_condition2 THEN sequence_of_statements2;
  ...
  WHEN search_conditionN THEN sequence_of_statementsN;
 [ELSE sequence_of_statementsN+1;]
END CASE;
```

The searched CASE statement has no selector. Also, its WHEN clauses contain search conditions that yield a Boolean value, not expressions that can yield a value of any type. An example follows:

```
CASE
  WHEN grade = 'A' THEN dbms_output.put_line('Excellent');
  WHEN grade = 'B' THEN dbms_output.put_line('Very Good');
  WHEN grade = 'C' THEN dbms_output.put_line('Good');
  WHEN grade = 'D' THEN dbms_output.put_line('Fair');
  WHEN grade = 'F' THEN dbms_output.put_line('Poor');
  ELSE dbms_output.put_line('No such grade');
```

```
END CASE;
```

Ex: with out FORALL

```
create table emp1
(
eid  number(4),
empname varchar2(20),
sal  number(5),
dept  number(2)
);

create or replace procedure p1(vdno emp.deptno%Type)
is
cursor c is select empno,ename,sal,deptno from emp where deptno=vdno;
type e_rec is record
                (
                empno emp.empno%type,
                ename emp.ename%type,
                sal     emp.sal%type,
                deptno     emp.deptno%type
                );
type e_arr  is table of c%rowtype index by pls_integer;
rec  e_arr;
a  int:=1;
begin
open c;
fetch c bulk collect into rec;
close c;
/* inserting row by row */
for x in 1..rec.count
loop
insert into emp1 values(rec(a).empno,rec(a).ename,rec(a).sal,rec(a).deptno);
a:=a+1;
end loop;
commit;
end p1;
```

```
FORALL indx IN 1 .. l_eligible_ids.COUNT
        UPDATE employees emp
           SET emp.salary =
                  emp.salary
                + emp.salary * increase_salary.increase_pct_in
         WHERE emp.employee_id = l_eligible_ids (indx);
   END increase_salary;
```

Ex: with FORALL

It is used to improve the performance while doing dml operation inside the loop.
Instead of for loop we are using FORALL keyword.

```
create table emp2
(
eid  number(4),
empname varchar2(20),
sal  number(5),
dept  number(2)
);


 DECLARE

       TYPE subset_rt IS RECORD
       ( empno    emp.empno%TYPE
       , ename    emp.ename%TYPE
       , hiredate emp.hiredate%TYPE
       , deptno   emp.deptno%TYPE );

       TYPE subset_aat IS TABLE OF subset_rt
         INDEX BY PLS_INTEGER;

      aa_subset subset_aat;

   BEGIN

      /* Some "source" data... */
      SELECT ROWNUM, owner, created, 20
      BULK   COLLECT INTO aa_subset
      FROM   all_objects
      WHERE  ROWNUM <= 10;

      /* Record-based insert and subset of columns... */
      FORALL i IN 1 .. aa_subset.COUNT
         INSERT INTO (SELECT empno, ename, hiredate, deptno FROM emp)
         VALUES aa_subset(i);

      DBMS_OUTPUT.PUT_LINE(TO_CHAR(SQL%ROWCOUNT) || ' rows inserted.');

   END;
   /
```

NOTE:

YOU CAN WATCH ORACLE VIDEOS FROM BELOW LINK.

https://www.youtube.com/results?search_query=oracle+videos+by+dinesh

THANK YOU
DINESH